

c o n f e r e n c e

p r o c e e d i n g s

**USENIX Workshop on
Smartcard Technology**

*Chicago, Illinois, USA
May 10–11, 1999*

Sponsored by

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
Email: office@usenix.org
WWW URL: <http://www.usenix.org>

The price is \$22 for members and \$28 for nonmembers.
Outside the U.S.A. and Canada, please add
\$12 per copy for postage (via air printed matter).

1999 © Copyright by The USENIX Association
All Rights Reserved

This volume is published as a collective work. Rights to individual papers remain with the author or the author's employer. Permission is granted for the noncommercial reproduction of the complete work for educational or research purposes. USENIX acknowledges all trademarks herein.

ISBN 1-880446-34-0

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

**Proceedings of the
USENIX Workshop on Smartcard Technology
(Smartcard '99)**

**May 10-11, 1999
Chicago, Illinois, USA**

Conference Organizers

Program Co-Chairs

Scott Guthery, *Microsoft Corporation*

Peter Honeyman, *CITI, University of Michigan*

Program Committee

Ron DeLange, *Lucent Technologies*

Michael Gallagher, *Datakey*

Peter Gutmann, *University of Auckland*

Dirk Husemann, *IBM*

Tim Jurgensen, *Schlumberger*

Piet Maclaine-Pont, *University of Groningen*

Markus Kuhn, *Cambridge University*

Jean-Marc Robert, *Gemplus*

Doug Tygar, *University of California*

The USENIX Association Staff

Contents

USENIX Workshop on Smartcard Technology (Smartcard '99)

May 10-11, 1999
Chicago, Illinois, USA

Message from the Program Chairs	v
Index of Authors	vi

Monday, May 10

Design

Feasibility of Smart Cards in Silicon-On-Insulator (SOI) Technology	1
<i>Amaury Nève, Denis Flandre, and Jean-Jacques Quisquater, Université Catholique de Louvain</i>	
Design Principles for Tamper-Resistant Smartcard Processors	9
<i>Oliver Kömmerling, Advanced Digital Security Research; Markus G. Kuhn, University of Cambridge</i>	
Which Security Policy for Multiapplication Smart Cards?	21
<i>Pierre Girard, Cryptography and Security R&D, GEMPLUS</i>	

Ciphers

Efficient Block Ciphers for Smartcards	29
<i>Joan Daemen, Proton World International; Vincent Rijmen, K.U.Leuven</i>	
PKCS #15—A Cryptographic-Token Information Format Standard	37
<i>Magnus Nyström, RSA Laboratories</i>	
Remotely Keyed Encryption Using Non-Encrypting Smart Cards	45
<i>Stefan Lucks and Rüdiger Weis, University of Mannheim</i>	

Authentication I

Smartcard Integration with Kerberos V5	51
<i>Naomaru Itoi and Peter Honeyman, University of Michigan, Ann Arbor</i>	
Mutual Authentication with Smart Cards	63
<i>Bastiaan Bakker, Delft University of Technology</i>	
Software License Management with Smart Cards	75
<i>Tuomas Aura, Helsinki University of Technology; Dieter Gollmann, Microsoft Research</i>	

Tuesday, May 11

Authentication II

Beyond Cryptographic Conditional Access	87
<i>David M. Goldschlag and David W. Kravitz, Divx</i>	

Providing Authentication to Messages Signed with a Smart Card in Hostile Environments	93
<i>Tage Stabell-Kulø, Ronny Arild, and Per Harald Myrvang, University of Tromsø</i>	
Authenticating Secure Tokens Using Slow Memory Access	101
<i>John Kelsey and Bruce Schneier, Counterpane Systems</i>	
Operating Systems	
SCFS: A UNIX Filesystem for Smartcards	107
<i>Naomaru Itoi, Peter Honeyman, and Jim Rees, University of Michigan, Ann Arbor</i>	
Secure Object Sharing in Java Card	119
<i>Michael Montgomery and Ksheerabdhi Krishna, Austin Product Center, Schlumberger</i>	
Object Lifetimes in Java Card	129
<i>Marcus Oestreicher, Zurich Research Laboratory, IBM Research Division; Ksheerabdhi Krishna, Austin Product Center, Schlumberger</i>	
A Personal Naming and Directory Service for Mobile Internet Users	139
<i>Alain Macaire and David Carlier, Gemplus Research Lab</i>	
Threats	
Investigations of Power Analysis Attacks on Smartcards	151
<i>Thomas S. Messerges and Ezzy A. Dabbish, Motorola Labs; Robert H. Sloan, University of Illinois at Chicago</i>	
Risks and Potentials of Using EMV for Internet Payments	163
<i>Els Van Herreweghen, IBM Zurich Research Laboratory; Uta Wille, Jelmoli Information Systems</i>	
Breaking Up Is Hard to Do: Modeling Security Threats for Smart Cards	175
<i>Bruce Schneier, Counterpane Systems; Adam Shostack, Netect, Inc.</i>	

Message from the Program Chairs

Welcome to the USENIX Workshop on Smartcard Technology, the first of its kind in North America. We are pleased to present 19 papers that detail the state of the art in smartcard-related research and development around the world.

The program—exceptionally strong for an inaugural workshop—reflects the international focus in smartcard research. It is no secret that deployment of smartcard systems in North America lags behind European activities, so it is no surprise that over half the papers in this proceedings originate in European laboratories. By coincidence, half of the Program Committee is European, half North American (and a lone Kiwi).

We thank the Program Committee for their diligent and timely reviews of the papers submitted. A total of 159 reviews were produced, averaging over a dozen reviews per committee member, so that each paper was reviewed by nearly half of the eleven-person team. While reviewing papers is usually thought of as a labor of love, it remains a labor, especially when scheduling requires that it be performed during the traditional winter holidays.

On behalf of the USENIX Association and the Program Committee, we welcome you to Chicago and hope you find the workshop challenging, rewarding, and, yes, fun.

Scott Guthery, *Microsoft Corporation*
Peter Honeyman, *CITI, University of Michigan*

Index of Authors

Arild, Ronny	93	Kulø, Tage Stabell-	93
Aura, Tuomas	75	Lucks, Stefan	45
Bakker, Bastiaan	63	Macaire, Alain	139
Carlier, David	139	Messerges, Thomas S.	151
Dabbish, Ezzy A.	151	Montgomery, Michael	119
Daemen, Joan	29	Myrvang, Per Harald	93
Flandre, Denis	1	Nève, Amaury	1
Girard, Pierre	21	Nyström, Magnus	37
Goldschlag, David M.	87	Oestreicher, Marcus	129
Gollmann, Dieter	75	Quisquater, Jean-Jacques	1
Herreweghen, Els Van	163	Rees, Jim	107
Honeyman, Peter	51, 107	Rijmen, Vincent	29
Itoi, Naomaru	51, 107	Schneier, Bruce	101, 175
Kelsey, John	101	Shostack, Adam	175
Kömmerling, Oliver	9	Sloan, Robert H.	151
Kravitz, David W.	87	Weis, Rüdiger	45
Krishna, Ksheerabधि	119, 129	Wille, Uta	163
Kuhn, Markus G.	9		

Feasibility of Smart Cards in Silicon-On-Insulator (SOI) Technology

Amaury Nève, Denis Flandre and Jean-Jacques Quisquater

*Université Catholique de Louvain
Microelectronics Laboratory
Place du Levant 3
B-1348 Louvain-la-Neuve, Belgium
neve@dice.ucl.ac.be*

Abstract

Applications involving smart cards have rapidly emerged since a few years. Up to now, chips are realized in conventional bulk technology. But as the need for performance rises, alternative technologies must be investigated. In this paper we study the feasibility of realizing the blocks for a smart card chip in Silicon-On-Insulator (SOI) technology. For most of the circuit blocks, SOI realization already exists and may be adapted for this application. However, we identified two circuits never fabricated in SOI: a charge pump and the random number generator. The charge pump has been realized in SOI and tested. A random signal source has also been realized. The circuit to create random bits, based on this source, is exposed.

1. Introduction

Applications involving smart cards have experienced a huge rise in recent years [1], [2], [3]. From simple memory cards at the origin, they evolved towards complex system-on-chip cards integrating memories, CPU, arithmetic co-processor and control logic. This opened new opportunities: smart cards can of course retain a huge amount of information compared to the magnetic strip cards, but they can also manage this information much more securely, using authentication and user identification procedures. The main improvement is the possibility to process the information directly on the card.

Chips for smart cards follow the general trends in

microelectronics [1]: towards small dimensions and towards low-power, low-voltage circuits. The performances required from the electronic circuits are more and more demanding [4]. Silicon-On-Insulator (SOI) is a very good candidate to fabricate high performance, low-power low-voltage VLSI circuits.

In the first part, we describe for which circuit blocks on the chip-card a realization already exists in SOI.

Secondly, two circuits, which have never been processed in SOI, are realized and tested: the charge pump and a random signal source that can be used in a random number generator.

The voltage multiplier or charge pump is used to generate a high voltage on-chip, in order to program non-volatile memories like EEPROM or Flash EEPROM.

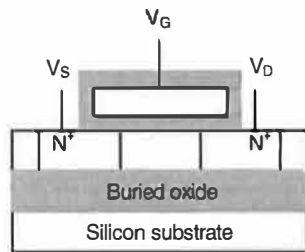
The random number generator generates a true random number that can be used during the authentication procedure. The intrinsic noise of transistors is used to generate a random signal. This random signal is sampled and transformed into a binary sequence. Statistical tests have been implemented to check whether the sequence has the characteristics of a true random sequence.

2. Silicon-On-Insulator technology

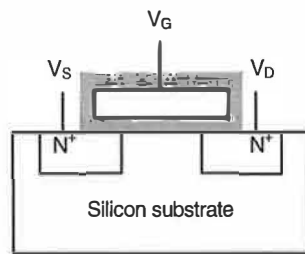
Silicon-On-Insulator transistors are fabricated in a small (~100 nm) layer of silicon, located on top of a silicon dioxide layer, called *buried oxide*. This oxide layer provides full dielectric isolation of the transistor, and

thus, most of the parasitic effects present in bulk silicon transistors are eliminated.

The structure of the SOI transistor is depicted in figure 1 (a), and is very similar to the structure of the bulk transistor (figure 1 (b)). The main difference is the presence of the buried oxide.



(a) SOI Transistor



(b) Bulk-Si Transistor

Figure 1 : Structure of a SOI MOS transistor (a) and a bulk-Si MOS transistor (b).

The presence of this buried oxide provides attractive properties to the SOI transistor.

The main advantages of SOI technology are summarized below [5], [6] :

- latchup of the parasitic PNP thyristor in CMOS circuits is eliminated,
- reduction of source and drain junction capacitances, which makes high speed operation possible,
- lower sensitivity to transient radiation effects,
- the fabrication technology is fully compatible with a conventional bulk CMOS process; the SOI process involves even less steps,
- higher integration density,
- high temperature operation.

If, in addition, the silicon film is made so thin that full depletion operation is achieved [6], the following advantages are also gained:

- improved subthreshold slope, and thus the possibility to lower the threshold voltage of the transistor without increasing the off-current,
- reduced body effect.

The most attractive properties for smart card applications are: lower operating voltage without loss of speed and the enhanced security. The latter is improved in three ways. Firstly, SOI technology allows the use of more compact layouts, which makes probing more difficult. Secondly, due to the lower power and current consumption, it will be less easy to measure the variations of these quantities. Finally, SOI circuits are recognized to operate well in harsh environments especially in high temperature and radiation environments.

3. Chip-card circuits

Based on the structure of the chip on a smart card (figure 2) [1], [7], [8], [9], we investigated the blocks already available in SOI technology and the compatibility of their performance with the smart card application.

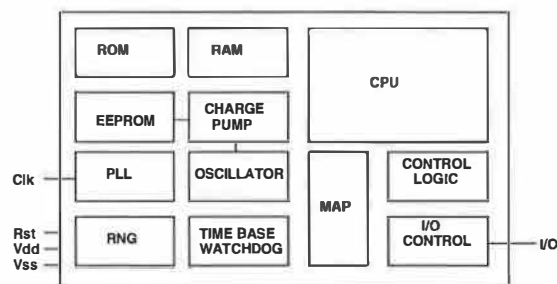


Figure 2 : Architecture of the chip for smart card applications [1], [8].

3.1 CPU

Many promising results have been published about processors in SOI. A research team from Motorola has made the first very low-power low-voltage CPU-core in SOI [10]. It showed superior performances compared to similar bulk realizations. For example, at 0.9 V supply

voltage, the SOI processor is twice as fast as a similar bulk processor.

A test version of the Strong ARM-110 has been processed and tested by Digital Equipment [11]. It showed a performance improvement of at least 20 % over the equivalent bulk circuit, and a gain of 30 % in power dissipation.

Other SOI logic circuits include Gate Arrays [12], [13] and ALU's [14]. In each case, the SOI circuits can operate faster than comparable bulk circuits, and can still operate at lower voltages.

These gains in performance, and especially the low-voltage operation, are significant regarding the smart card application.

3.2 Clock generation circuits

In a smart card, a clock signal is provided by the outside world and is supplied to an on-chip clock regeneration circuit, in order to stabilize the signal and to prevent clock signal manipulations. The circuit can be implemented as a Phase-Locked Loop (PLL). Such circuits have been made in SOI. For example, NTT has developed a PLL that operates at 2 V supply voltage, up to a frequency of 2 GHz [15], [16]. This shows that it is possible to conceive low-voltage clock regeneration circuits in SOI for the smart card application.

3.3 Memories

Three types of memory are present on the smart card chip: ROM, RAM and non-volatile memory like EEPROM or Flash EEPROM [7].

The ROM is programmed by mask during device fabrication and no particular problem is encountered. One example of SOI realization can be found in the CPU core described above [10].

The RAM of smart cards is currently Static RAM (SRAM). The main reason is the possibility to use a power-saving mode [9]: when the CPU stays in sleep mode, the clock is fixed to the high or the low level permanently. Whereas a Dynamic RAM (DRAM) needs to be periodically refreshed, the SRAM doesn't need this and the presence of the supply voltage is sufficient to

retain the information.

A SRAM cell occupies more die area than a DRAM cell (6 transistors vs. one transistor and a tiny capacitor), but this is compensated by the absence of refresh circuit for small memories (36...256 bytes). Several manufacturers master well the fabrication of SOI-SRAM [17], [18] and SOI-DRAM [19], [20] for some years now, with speed and power performances superior to conventional bulk silicon memories.

Most of the chip cards carry their user-specific information in an EEPROM or Flash EEPROM memory. Although the research in the field of the non-volatile memories in SOI is more recent, some realizations can be found. Martignone [21] described an EEPROM-based cell realized in the SOI CMOS technology of the UCL Microelectronics Laboratory. Gogl et al. realized a single-polysilicon EEPROM-cell in SOI technology for high temperature applications [22]. Two Flash EEPROM cells were described, one [23] fabricated in the UCL Microelectronics Laboratory, the other [24] fabricated by National Semiconductor.

4. New circuits in SOI

From the previous investigation, we identified two circuits which have never been realized in SOI : the charge pump and the random number generator. In the present work, a charge pump has been realized in the SOI-CMOS 2 μm technology of the UCL Microelectronics Laboratory. The architecture of a random number generator is also proposed. It is based on a random signal generator, realized in SOI, and some additional components to transform the random signal in a stream of random bits. These components are external discrete components in the present implementation, but could easily be integrated in SOI technology.

4.1 The charge pump

Charge pump circuits, or voltage multipliers, are required to program the non-volatile memory. EEPROM or Flash EEPROM cells need a high voltage to be programmed, formerly 20 V, now towards 5 V. It can be generated on-chip from the lower supply voltage with a

charge pump circuit. Most of these circuits are based on the Dickson charge pump [25], [26] (figure 3).

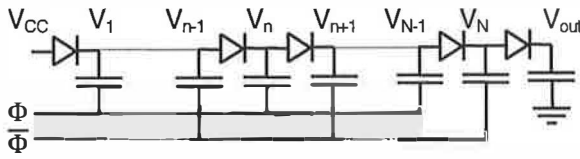


Figure 3 : Dickson voltage multiplier [25].

The electric charges are pumped from node to node, from the supply to the output node. The pumping is achieved by charging and discharging the capacitors, under influence of the complementary clock signals Φ and $\bar{\Phi}$. The voltage increases from node to node, to reach the final voltage on the output node.

In CMOS technology, the diodes are replaced by MOS transistors, which have their gate and drain in short-circuit. The circuit we have realized is shown in figure 4.

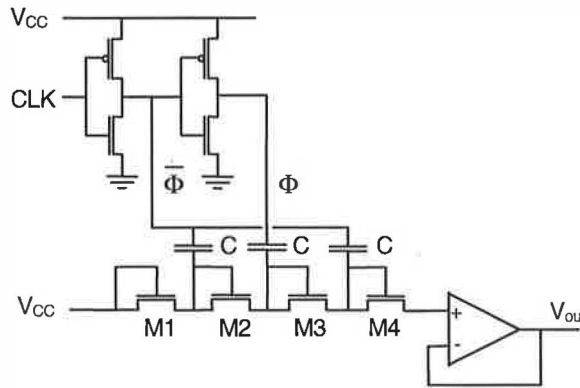


Figure 4 : Schematic of the charge pump circuit realized in the Microelectronics Laboratory.

The external clock signal (CLK) is fed to the MOS diodes through two inverters, to get complementary signals. An amplifier in follower configuration has been placed at the output terminal in our test implementation in order to reduce the coupling between the load capacitance and the output voltage during the measurements. This would not be present in a practical EEPROM architecture.

The experimental results are presented in figure 5 and compared to the theoretical prediction of an analytical modeling.

For low supply voltages, the experimental points follow the theoretical curve fairly well. For supply voltages above 1.4 V, a saturation phenomenon can be observed. This is due to the limitation of the dynamic range of the follower-amplifier placed at the output.

It can be observed that the supply voltage is multiplied by a factor of three, which was the target of our

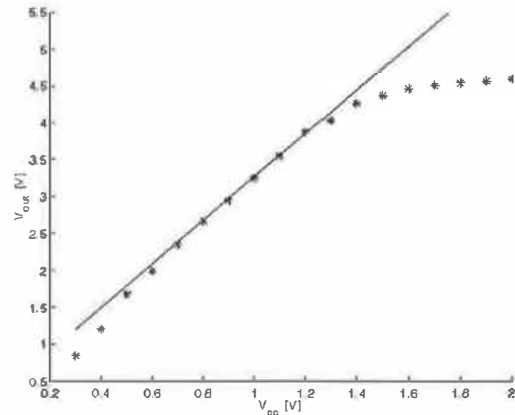


Figure 5 : Experimental (*) and theoretical (-) results for the charge pump circuit. This figure presents the output voltage V_{out} vs. supply voltage V_{cc} .

application. A programming voltage of 5 V could be obtained from a supply voltage of 1.8 V in a practical EEPROM implementation (without measurement output amplifier).

4.2 The random number generator

4.2.1 Design of the generator

The random number generator in the smart card is used during the authentication procedure [9]. The terminal asks a random number to the card. The smart card encrypts it, and sends it to the terminal. If the terminal is able to decipher the number, it is authenticated by the card. Most of the generators used in current smart cards are based on deterministic algorithms expanding a random seed, thus producing «pseudo-random numbers». But the security of the application is not necessarily guaranteed [27].

In this work we propose to develop a hardware random number generator, producing true random numbers. In the literature, very few realizations of integrated random

number generators are described [28]. Some examples of random signal sources are: the frequency instability of an oscillator [28], the noise of semiconductors [29], [30], the time between two radioactive emissions [31], the number of charges in a MOS capacitor structure [32].

We use the intrinsic noise of transistors as random signal. It presents the advantage of being present in the integrated circuit itself. It is easy to use in low voltage applications.

A single-stage Operational Transconductance Amplifier (OTA) is used to produce noise (figure 6).

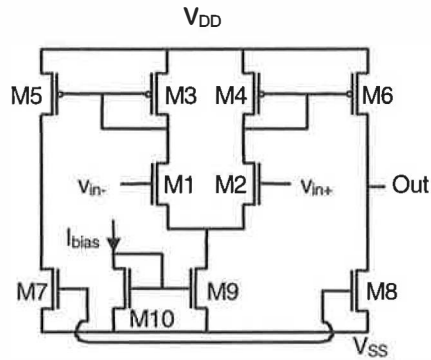


Figure 6 : Single-stage OTA used to produce noise.

This OTA is designed to produce a high level of noise at its output. The most important contribution comes from the input differential pair M1 and M2 [33].

The noise source is integrated in a more global architecture, presented in figure 7.

The experimental circuit is composed of two parts: the random signal generator, integrated in SOI, represented in the box in figure 7, and some external circuitry made

up of discrete components.

The random signal generator uses two independent noise sources. Their outputs provide a random signal of a few microvolts of amplitude, set around a fixed DC level.

The two signals are fed into a comparator. Provided that the DC levels are the same on each comparator input, the comparator will switch randomly from one level to the other. The produced signal will be an analog random signal. The bandpass filter (*B.P. filter* in figure 7) selects the frequency band, and thus eliminates parasitic signals at low and high frequencies.

The random analog signal is then sampled and compared to a reference value. This reference value must be the mean of the random signal. The clock signal used for the sampling determines the speed at which the random bits will be produced.

4.2.2 Experimental results

The output signal of the SOI random signal generator is shown in figure 8. The random signal is modulated by a low frequency signal. The band-pass filter will eliminate the latter.

Sequences of random bits were produced using this random signal in the circuit described above. Menezes et al. ([34], Chapter 5, pp. 169-190) propose 5 statistical tests to check for the randomness of the sequences: the equidistribution test, the serial test, the gap test, the poker test, the runs test and the autocorrelation test.

When applying the tests to the produced sequences, only 11 % of them pass the 5 tests successfully. The other sequences present a bias, that is a clear preference for «0»'s or «1»'s. This seems to be a general phenomenon affecting hardware random number generators [34].

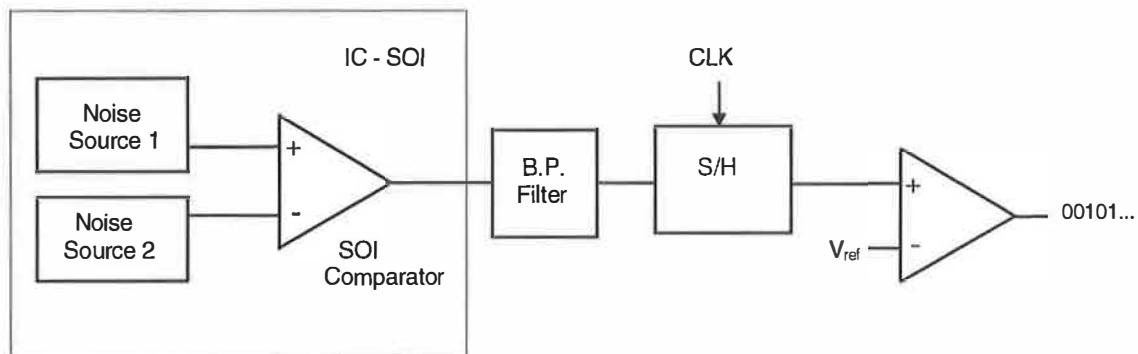


Figure 7 : Complete architecture of the random number generator.

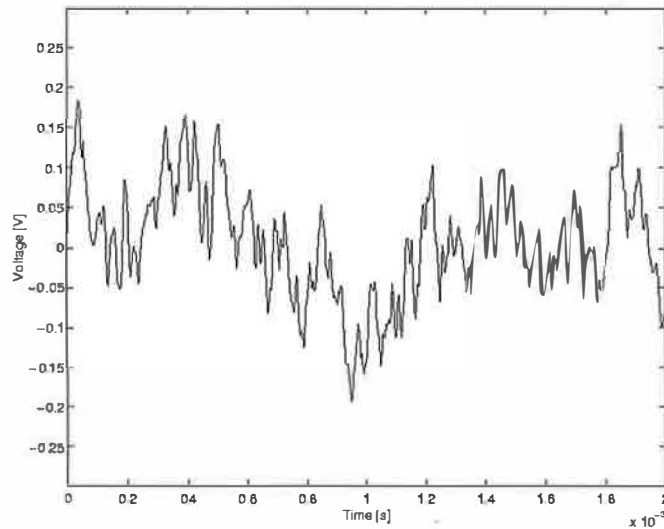


Figure 8 : Output voltage of the SOI random signal generator vs. time.

It is possible to de-skew the sequences by using de-skewing algorithms. Here we used Von Neumann's method and the parity bit method [27]. In that case, 96 % of the produced sequences pass the 5 statistical tests.

5. Conclusion

The objective of this work was to demonstrate the feasibility of realizing the smart card IC in Silicon-On-Insulator (SOI) technology. In the first part, we established the global architecture, and we checked in the literature which circuit blocks have already been demonstrated in SOI. For most of the circuit parts, a realization exists, and can be adapted for use in the smart card.

We identified two circuit blocks, never realized in SOI up to now to our knowledge : the charge pump, and a random number generator.

The charge pump has been realized and tested.

A new architecture for a random number generator is proposed. A random signal generator has been processed in SOI and is used to produce random numbers. In the next version of the generator, a regulation circuit for the DC levels must be included. This will enhance the immunity to external effects (temperature, electromagnetic waves).

6. References

- [1] Fancher C.H., *In your pocket : smart cards*, in IEEE Spectrum, vol. 34, n° 2, february 1997, pp. 47-53.
- [2] Jarvis C.R., *Beyond the Phone Card : Emerging Smart Card Opportunities*, in GEC Review, pp. 131-137, vol. 12, n° 3, 1997.
- [3] Quisquater J.-J., *The adolescence of smart cards*, in Future Generation Computer Systems, n° 13, 1997, pp. 3-7.
- [4] <http://www.dice.ucl.ac.be/~dhem/cascade>
- [5] Colinge J.-P., *Performances of Low-Voltage, Low-Power SOI CMOS Technology*, in Proceedings 21st Int. Conference on Microelectronics, vol. 1, september 1997, pp. 229-235.
- [6] Colinge J.-P., *Silicon-on-Insulator Technology: Materials to VLSI*, 2nd edition, Kluwer Academic Publishers, 1997.
- [7] Guillou L.C., Ugon M., Quisquater J.-J., *The Smart Card. A standardized Security Device Dedicated to Public Cryptology*, in Contemporary Cryptology, edited by Simmons G.J., IEEE Press, 1992, pp. 561-613.
- [8] Motorola, *Technical Summary, MSC0501, 8-bit microcontroller with Modular Arithmetic Processor*, 1997.
- [9] Rankl W., Effing W., *Smartcard Handbook*, Wiley, 1997.

- [10] Huang W.M., Papworth K., Racanelli M., John J.P., Foerstner J., Shin H.C., Park H., Hwang B.Y., Wetteroth, Hong S., Shin H., Wilson S., Cheng S., *TFSOI CMOS Technology for sub-1V Microcontroller Circuits*, in IEDM, 1995, pp. 59-62.
- [11] Mistry K., Grula G., Sleight J., Bair L., Stephany R., Flatley R., Skerry P., *A 2.0 V, 0.35 μ m Partially Depleted SOI-CMOS Technology*, in IEDM, 1997, pp. 583-586.
- [12] Kimio U. et al., *A CAD-Compatible SOI/CMOS Gate Array having Body Fixed Partially Depleted Transistors*, in IEEE International Solide-State Circuits Conference, 1997, pp. 288-289.
- [13] Masayuki et al., *0.25 μ m CMOS/SIMOX Gate Array LSI*, in IEEE International Solide-State Circuits Conference, 1996, pp. 86-87.
- [14] Tsuneaki F. et al., *A 0.5 V 200 MHz 1-Stage 32b ALU using a Body Bias Controlled SOI Pass-Gate Logic*, in IEEE International Solide-State Circuits Conference, 1997, pp. 286-287.
- [15] Fujishima M., Asada K., Omura Y., Izumi K., *Low Power $\frac{1}{2}$ Frequency Dividers Using 0.1 μ m CMOS Circuits Built with Ultrathin SIMOX Substrates*, in IEEE Journal of Solide-State Circuits, vol. 28, n° 4, april 1993, pp. 510 - 512.
- [16] Kado Y., Suzuki M., Koike K., Omura Y., Izumi K., *A 1GHz/0.9 mW CMOS/SIMOX Divide-by-128/129 Dual Modulus Prescaler Using a Divide-by-2/3 Synchronous Counter*, in IEEE Journal of Solide-State Circuits, vol. 28, n° 4, april 1993, pp. 513-517.
- [17] Kikuchi T., Onishi Y., Hashimoto T., Yoshida E., Yamaguchi H., Wada S., Tamba N., Watanbe K., Tamaki Y., Ikeda T., *A 0.35 μ m ECL-CMOS Process Technology on SOI for 1 ns Mega-bits SRAM's with 40 ps Gate Array*, IEDM, 1995, pp. 923-926.
- [18] Lu H., Yee E., Hite L., Houston T., Sheu Y.D., Rajgopal R., Shen C.C., Hwang J.M., Pollack G., *A 1-M bit SRAM on SIMOX material*, in Proceedings 1993 IEEE International SOI Conference, 1993, pp. 182-183.
- [19] Oashi T. et al., *16 Mb DRAM/SOI Technologies for Sub 1 V Operation*, in IEDM, 1996, pp. 609-612.
- [20] Koh et al., *1 Giga Bit SOI DRAM with Fully Bulk Compatible Process and Body-Contacted SOI MOSFET Structure*, in IEDM, 1997, pp. 579-582.
- [21] Martignone R., *Analog Single-Poly Floating-Gate Memories for Neural Network Applications*, Travail de fin d'études, Laboratoire de Microélectronique, UCL, 1996.
- [22] Gogl D., Burbach G., Fiedler H.-L., Verbeck M., Zimmermann C., *A Single-Poly EEPROM Cell in SIMOX technology for High-Temperature Applications up to 250 °C*, in IEEE Electron Device Letters, vol. 18, n°11, nov. 1997, pp. 541-543.
- [23] Zaleski A., Ioannou D.E., Flandre D., Colinge J.P., *Design and performance of a new Flash EEPROM on SOI (SIMOX) substrates*, in Proceedings 1994 IEEE International SOI Conference, oct. 1994, pp. 13-14.
- [24] Chi M.-H., Bergemont A., *Programming and erase with floating-body for high density low voltage Flash EEPROM fabricated on SOI wafers*, in Proceeding 1995 IEEE International SOI Conference, oct. 1995, pp. 129-130.
- [25] Dickson J.F., *On-Chip High Voltage Generator in NMOS Integrated Circuits Using an Improved Voltage Multiplier Technique*, in IEEE Journal of Solide-State Circuits, vol. SC-11, n° 3, june 1976, pp. 374-378.
- [26] Witters J.S., Groeseneken G., Maes H.E., *Analysis and Modeling of On-Chip High Voltage Generator Circuits for Use in EEPROM Circuits*, in IEEE Journal of Solide-State Circuits, vol. 24, n° 5, october 1989, pp. 1327-1380.
- [27] <http://ds.internic.net/rfc/rfc1750>
- [28] Fairfield R.C., Mortenson R. L., Coulthart K.B., *An LSI Random Number Generator*, in Advances in Cryptology, Proceedings of CRYPTO '84, pp. 203-230.
- [29] <http://www.protego.se>
- [30] <http://shell.rmi.net/~comscire>
- [31] <http://www.fourmilab.ch/hotbits>
- [32] Agnew G.B., *Random Sources for Cryptographic systems*, in Advances in Cryptology, EUROCRYPT '87, pp. 77-81.
- [33] Dillies J., *Etude, mesures, réalisation de dispositifs à haut rapport signal sur bruit en technologie SOI*, Travail de fin d'études, Laboratoire de Microélectronique, UCL, 1995.
- [34] Menezes A.J., van Oorschot P.C., Vanstone S.A., *Handbook of Applied Cryptography*, CRC Press, 1996.

Design Principles for Tamper-Resistant Smartcard Processors

Oliver Kömmerling

Markus G. Kuhn

*Advanced Digital
Security Research
Mühlstraße 7
66484 Riedelberg
Germany
ok@adsr.de*

*University of Cambridge
Computer Laboratory
Pembroke Street
Cambridge CB2 3QG
United Kingdom
mgk25@cl.cam.ac.uk*

Abstract

We describe techniques for extracting protected software and data from smartcard processors. This includes manual microprobing, laser cutting, focused ion-beam manipulation, glitch attacks, and power analysis. Many of these methods have already been used to compromise widely-fielded conditional-access systems, and current smartcards offer little protection against them. We give examples of low-cost protection concepts that make such attacks considerably more difficult.

1 Introduction

Smartcard piracy has become a common occurrence. Since around 1994, almost every type of smartcard processor used in European, and later also American and Asian, pay-TV conditional-access systems has been successfully reverse engineered. Compromised secrets have been sold in the form of illicit clone cards that decrypt TV channels without revenue for the broadcaster. The industry has had to update the security processor technology several times already and the race is far from over.

Smartcards promise numerous security benefits. They can participate in cryptographic protocols, and unlike magnetic stripe cards, the stored data can be protected against unauthorized access. However, the strength of this protection seems to be frequently overestimated.

In Section 2, we give a brief overview on the most important hardware techniques for breaking into smartcards. We aim to help software engineers without a background in modern VLSI test techniques in getting a realistic impression of how physical tampering works and what it costs. Based on our observations of what makes these attacks particularly easy, in Section 3 we discuss various ideas

for countermeasures. Some of these we believe to be new, while others have already been implemented in products but are either not widely used or have design flaws that have allowed us to circumvent them.

2 Tampering Techniques

We can distinguish four major attack categories:

- **Microprobing** techniques can be used to access the chip surface directly, thus we can observe, manipulate, and interfere with the integrated circuit.
- **Software attacks** use the normal communication interface of the processor and exploit security vulnerabilities found in the protocols, cryptographic algorithms, or their implementation.
- **Eavesdropping** techniques monitor, with high time resolution, the analog characteristics of all supply and interface connections and any other electromagnetic radiation produced by the processor during normal operation.
- **Fault generation** techniques use abnormal environmental conditions to generate malfunctions in the processor that provide additional access.

All microprobing techniques are *invasive attacks*. They require hours or weeks in a specialized laboratory and in the process they destroy the packaging. The other three are *non-invasive attacks*. After we have prepared such an attack for a specific processor type and software version, we can usually reproduce it within seconds on another card of the same type. The attacked card is not physically harmed and the equipment used in the attack can usually be disguised as a normal smartcard reader.

Non-invasive attacks are particularly dangerous in some applications for two reasons. Firstly, the

owner of the compromised card might not notice that the secret keys have been stolen, therefore it is unlikely that the validity of the compromised keys will be revoked before they are abused. Secondly, non-invasive attacks often scale well, as the necessary equipment (e.g., a small DSP board with special software) can usually be reproduced and updated at low cost.

The design of most non-invasive attacks requires detailed knowledge of both the processor and software. On the other hand, invasive microprobing attacks require very little initial knowledge and usually work with a similar set of techniques on a wide range of products. Attacks therefore often start with invasive reverse engineering, the results of which then help to develop cheaper and faster non-invasive attacks. We have seen this pattern numerous times on the conditional-access piracy market.

Non-invasive attacks are of particular concern in applications where the security processor is primarily required to provide *tamper evidence*, while invasive attacks violate the *tamper-resistance* characteristics of a card [1]. Tamper evidence is of primary concern in applications such as banking and digital signatures, where the validity of keys can easily be revoked and where the owner of the card has already all the access that the keys provide anyway. Tamper resistance is of importance in applications such as copyright enforcement, intellectual property protection, and some electronic cash schemes, where the security of an entire system collapses as soon as a few cards are compromised.

To understand better which countermeasures are of practical value, we first of all have to understand the techniques that pirates have used so far to break practically all major smartcard processors on the market. In the next section, we give a short guided tour through a typical laboratory of a smartcard pirate.

2.1 Invasive Attacks

2.1.1 Depackaging of Smartcards

Invasive attacks start with the removal of the chip package. We heat the card plastic until it becomes flexible. This softens the glue and the chip module can then be removed easily by bending the card. We cover the chip module with 20–50 ml of fuming nitric acid heated to around 60 °C and wait for the black epoxy resin that encapsulates the silicon die to completely dissolve (Fig. 1). The procedure should preferably be carried out under very dry conditions, as the presence of water could corrode exposed aluminium interconnects. The chip is then washed with

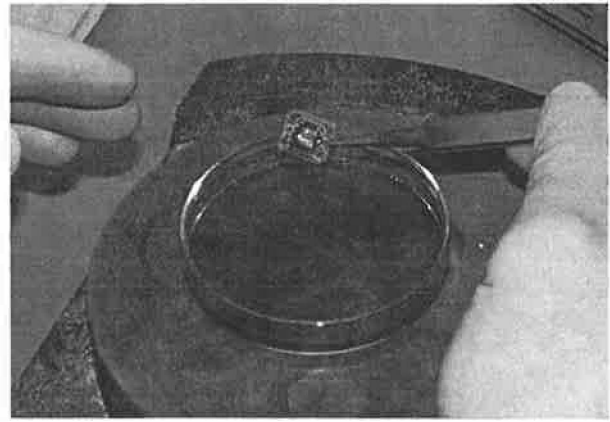


Figure 1: Hot fuming nitric acid ($> 98\% \text{HNO}_3$) dissolves the package without affecting the chip.



Figure 2: The depackaged smartcard processor is glued into a test package, whose pins are then connected to the contact pads of the chip with fine aluminium wires in a manual bonding machine.

acetone in an ultrasonic bath, followed optionally by a short bath in deionized water and isopropanol. We remove the remaining bonding wires with tweezers, glue the die into a test package, and bond its pads manually to the pins (Fig. 2). Detailed descriptions of these and other preparation techniques are given in [2, 3].

2.1.2 Layout Reconstruction

The next step in an invasive attack on a new processor is to create a map of it. We use an optical microscope with a CCD camera to produce several meter large mosaics of high-resolution photographs of the chip surface. Basic architectural structures, such as data and address bus lines, can be identified quite quickly by studying connectivity patterns

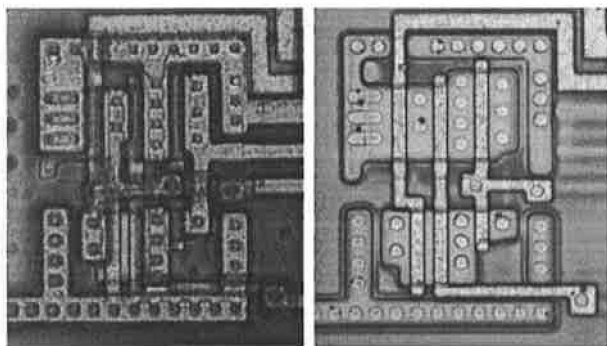


Figure 3: Left: CMOS AND gate imaged by a confocal microscope. Right: same gate after removal of metal layer (HF wet etching). Polysilicon interconnects and diffusion areas are now fully visible.

and by tracing metal lines that cross clearly visible module boundaries (ROM, RAM, EEPROM, ALU, instruction decoder, etc.). All processing modules are usually connected to the main bus via easily recognizable latches and bus drivers. The attacker obviously has to be well familiar with CMOS VLSI design techniques and microcontroller architectures, but the necessary knowledge is easily available from numerous textbooks [4, 5, 6, 7].

Photographs of the chip surface show the top metal layer, which is not transparent and therefore obscures the view on many structures below. Unless the oxide layers have been planarized, lower layers can still be recognized through the height variations that they cause in the covering layers. Deeper layers can only be recognized in a second series of photographs after the metal layers have been stripped off, which we achieve by submerging the chip for a few seconds in hydrofluoric acid (HF) in an ultrasonic bath [2]. HF quickly dissolves the silicon oxide around the metal tracks and detaches them from the chip surface. HF is an extremely dangerous substance and safety precautions have to be followed carefully when handling it.

Figure 3 demonstrates an optical layout reconstruction of a NAND gate followed by an inverter. These images were taken with a confocal microscope (Zeiss Axiotron-2 CSM), which assigns different colors to different focal planes (e.g., metal=blue, polysilicon=green) and thus preserves depth information [8]. Multilayer images like those shown in Fig. 3 can be read with some experience almost as easily as circuit diagrams. These photographs help us in understanding those parts of the circuitry that are relevant for the planned attack.

If the processor has a commonly accessible standard architecture, then we have to reconstruct the

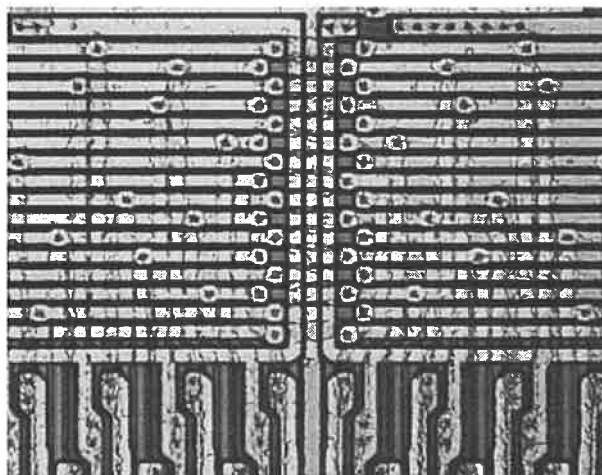


Figure 4: The vias in this structure found in a ST16F48A form a permutation matrix between the memory readout column lines and the 16:1 demultiplexer. The applied mapping remains clearly visible.

layout only until we have identified those bus lines and functional modules that we have to manipulate to access all memory values. More recently, designers of conditional-access smartcards have started to add proprietary cryptographic hardware functions that forced the attackers to reconstruct more complex circuitry involving several thousand transistors before the system was fully compromised. However, the use of standard-cell ASIC designs allows us to easily identify logic gates from their diffusion area layout, which makes the task significantly easier than the reconstruction of a transistor-level netlist.

Some manufacturers use non-standard instruction sets and bus-scrambling techniques in their security processors. In this case, the entire path from the EEPROM memory cells to the instruction decoder and ALU has to be examined carefully before a successful disassembly of extracted machine code becomes possible. However, the attempts of bus scrambling that we encountered so far in smartcard processors were mostly only simple permutations of lines that can be spotted easily (Fig. 4).

Any good microscope can be used in optical VLSI layout reconstruction, but confocal microscopes have a number of properties that make them particularly suited for this task. While normal microscopes produce a blurred image of any plane that is out of focus, in confocal scanning optical microscopes, everything outside the focal plane just becomes dark [8]. Confocal microscopes also provide better resolution and contrast. A chromatic lens in the system can make the location of the focal plane wavelength dependent, such that under white light different layers

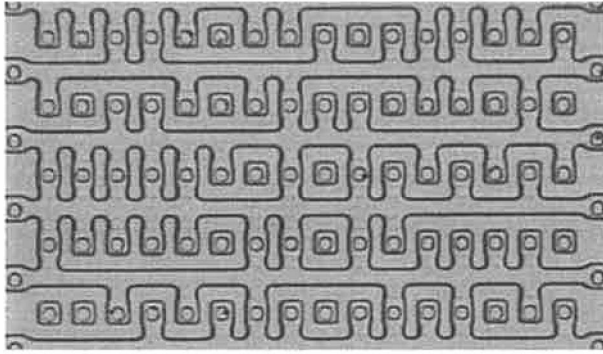


Figure 5: The data of this NOR ROM becomes clearly visible when the covering metal and polysilicon access lines plus the surrounding field oxide have been removed (HF wet etching). The image shows 16×10 bits in an ST16xyz. Every bit is represented by either a present or missing diffusion layer connection.

of the chip will appear simultaneously, but in different colors.

Automatic layout reconstruction has been demonstrated with scanning electron microscopy [9]. We consider confocal microscopy to be an attractive alternative, because we do not need a vacuum environment, the depth information is preserved, and the option of oil immersion allows the hiding of unevenly removed oxide layers. With UV microscopy, even chip structures down to $0.1 \mu\text{m}$ can be resolved.

With semiautomatic image-processing methods, significant portions of a processor can be reverse engineered within a few days. The resulting polygon data can then be used to automatically generate transistor and gate-level netlists for circuit simulations.

Optical reconstruction techniques can also be used to read ROM directly. The ROM bit pattern is stored in the diffusion layer, which leaves hardly any optical indication of the data on the chip surface. We have to remove all covering layers using HF wet etching, after which we can easily recognize the rims of the diffusion regions that reveal the stored bit pattern (Fig. 5).

Some ROM technologies store bits not in the shape of the active area but by modifying transistor threshold voltages. In this case, additional dopant-selective staining techniques have to be applied to make the bits visible (Fig. 6). Together with an understanding of the (sometimes slightly scrambled, see Fig. 4) memory-cell addressing, we obtain disassembler listings of the entire ROM content. Again, automated processing techniques can be used to extract the data from photos, but we also know cases

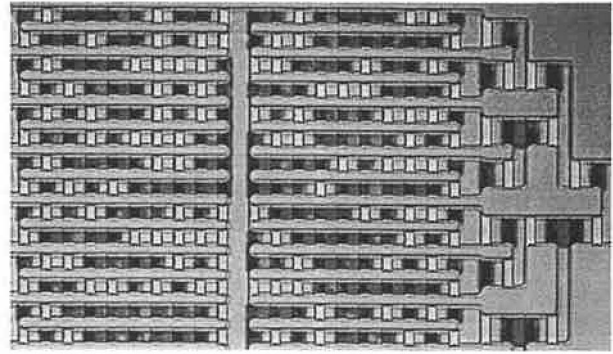


Figure 6: The implant-mask layout of a NAND ROM can be made visible by a dopant-selective crystallographic etch (Dash etch and [2]). This image shows 16×14 bits plus parts of the row selector of a ROM found on an MC68HC05SC2x CPU. The threshold voltage of 0-bit p-channel transistors (stained dark here) was brought below 0 V through ion implantation.

where an enthusiastic smartcard hacker has reconstructed several kilobytes of ROM manually.

While the ROM usually does not contain any cryptographic key material, it does often contain enough I/O, access control, and cryptographic routines to be of use in the design of a non-invasive attack.

2.1.3 Manual Microprobing

The most important tool for invasive attacks is a microprobing workstation. Its major component is a special optical microscope (e.g., Mitutoyo FS-60) with a working distance of at least 8 mm between the chip surface and the objective lens. On a stable platform around a socket for the test package, we install several micropositioners (e.g., from Karl Suss, Micromanipulator, or Wentworth Labs), which allow us to move a probe arm with submicrometer precision over a chip surface. On this arm, we install a “cat whisker” probe (e.g., Picoprobe T-4-10). This is a metal shaft that holds a $10 \mu\text{m}$ diameter and 5 mm long tungsten-hair, which has been sharpened at the end into a $< 0.1 \mu\text{m}$ tip. These elastic probe hairs allow us to establish electrical contact with on-chip bus lines without damaging them. We connect them via an amplifier to a digital signal processor card that records or overrides processor signals and also provides the power, clock, reset, and I/O signals needed to operate the processor via the pins of the test package.

On the depackaged chip, the top-layer aluminium interconnect lines are still covered by a passivation

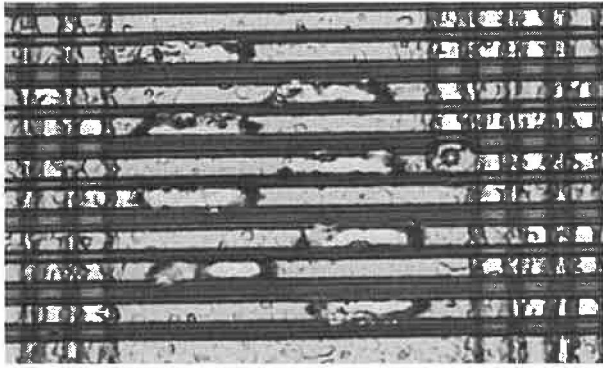


Figure 7: This image shows 9 horizontal bus lines on a depackaged smartcard processor. A UV laser (355 nm, 5 ns) was used to remove small patches of the passivation layer over the eight data-bus lines to provide for microprobing access.

layer (usually silicon oxide or nitride), which protects the chip from the environment and ion migration. On top of this, we might also find a polyimide layer that was not entirely removed by HNO_3 but which can be dissolved with ethylenediamine. We have to remove the passivation layer before the probes can establish contact. The most convenient depassivation technique is the use of a laser cutter (e.g., from New Wave Research).

The UV or green laser is mounted on the camera port of the microscope and fires laser pulses through the microscope onto rectangular areas of the chip with micrometer precision. Carefully dosed laser flashes remove patches of the passivation layer. The resulting hole in the passivation layer can be made so small that only a single bus line is exposed (Fig. 7). This prevents accidental contacts with neighbouring lines and the hole also stabilizes the position of the probe and makes it less sensitive to vibrations and temperature changes.

Complete microprobing workstations cost tens of thousands of dollars, with the more luxurious versions reaching over a hundred thousand US\$. The cost of a new laser cutter is roughly in the same region.

Low-budget attackers are likely to get a cheaper solution on the second-hand market for semiconductor test equipment. With patience and skill it should not be too difficult to assemble all the required tools for even under ten thousand US\$ by buying a second-hand microscope and using self-designed micropositioners. The laser is not essential for first results, because vibrations in the probing needle can also be used to break holes into the passivation.

2.1.4 Memory Read-out Techniques

It is usually not practical to read the information stored on a security processor directly out of each single memory cell, except for ROM. The stored data has to be accessed via the memory bus where all data is available at a single location. Microprobing is used to observe the entire bus and record the values in memory as they are accessed.

It is difficult to observe all (usually over 20) data and address bus lines at the same time. Various techniques can be used to get around this problem. For instance we can repeat the same transaction many times and use only two to four probes to observe various subsets of the bus lines. As long as the processor performs the same sequence of memory accesses each time, we can combine the recorded bus subset signals into a complete bus trace. Overlapping bus lines in the various recordings help us to synchronize them before they are combined.

In applications such as pay-TV, attackers can easily replay some authentic protocol exchange with the card during a microprobing examination. These applications cannot implement strong replay protections in their protocols, because the transaction counters required to do this would cause an NVRAM write access per transaction. Some conditional-access cards have to perform over a thousand protocol exchanges per hour and EEPROM technology allows only 10^4 – 10^6 write cycles during the lifetime of a storage cell. An NVRAM transaction counter would damage the memory cells, and a RAM counter can be reset by the attacker easily by removing power. Newer memory technologies such as FFERAM allow over 10^9 write cycles, which should solve this problem.

Just replaying transactions might not suffice to make the processor access all critical memory locations. For instance, some banking cards read critical keys from memory only after authenticating that they are indeed talking to an ATM. Pay-TV card designers have started to implement many different encryption keys and variations of encryption algorithms in every card, and they switch between these every few weeks. The memory locations of algorithm and key variations are not accessed by the processor before these variations have been activated by a signed message from the broadcaster, so that passive monitoring of bus lines will not reveal these secrets to an attacker early.

Sometimes, hostile bus observers are lucky and encounter a card where the programmer believed that by calculating and verifying some memory checksum after every reset the tamper-resistance

could somehow be increased. This gives the attacker of course easy immediate access to all memory locations on the bus and simplifies completing the read-out operation considerably. Surprisingly, such memory integrity checks were even suggested in the smartcard security literature [10], in order to defeat a proposed memory rewrite attack technique [11]. This demonstrates the importance of training the designers of security processors and applications in performing a wide range of attacks before they start to design countermeasures. Otherwise, measures against one attack can far too easily backfire and simplify other approaches in unexpected ways.

In order to read out all memory cells without the help of the card software, we have to abuse a CPU component as an address counter to access all memory cells for us. The program counter is already incremented automatically during every instruction cycle and used to read the next address, which makes it perfectly suited to serve us as an address sequence generator [12]. We only have to prevent the processor from executing jump, call, or return instructions, which would disturb the program counter in its normal read sequence. Tiny modifications of the instruction decoder or program counter circuit, which can easily be performed by opening the right metal interconnect with a laser, often have the desired effect.

2.1.5 Particle Beam Techniques

Most currently available smartcard processors have feature sizes of 0.5–1 μm and only two metal layers. These can be reverse-engineered and observed with the manual and optical techniques described in the previous sections. For future card generations with more metal layers and features below the wavelength of visible light, more expensive tools additionally might have to be used.

A focused ion beam (FIB) workstation consists of a vacuum chamber with a particle gun, comparable to a scanning electron microscope (SEM). Gallium ions are accelerated and focused from a liquid metal cathode with 30 kV into a beam of down to 5–10 nm diameter, with beam currents ranging from 1 pA to 10 nA. FIBs can image samples from secondary particles similar to a SEM with down to 5 nm resolution. By increasing the beam current, chip material can be removed with the same resolution at a rate of around $0.25 \mu\text{m}^3 \text{nA}^{-1} \text{s}^{-1}$ [13]. Better etch rates can be achieved by injecting a gas like iodine via a needle that is brought to within a few hundred micrometers from the beam target. Gas molecules settle down on the chip surface and react with removed material to

form a volatile compound that can be pumped away and is not redeposited. Using this gas-assisted etch technique, holes that are up to 12 times deeper than wide can be created at arbitrary angles to get access to deep metal layers without damaging nearby structures. By injecting a platinum-based organometallic gas that is broken down on the chip surface by the ion beam, platinum can be deposited to establish new contacts. With other gas chemistries, even insulators can be deposited to establish surface contacts to deep metal without contacting any covering layers.

Using laser interferometer stages, a FIB operator can navigate blindly on a chip surface with 0.15 μm precision, even if the chip has been planarized and has no recognizable surface structures. Chips can also be polished from the back side down to a thickness of just a few tens of micrometers. Using laser-interferometer navigation or infrared laser imaging, it is then possible to locate individual transistors and contact them through the silicon substrate by FIB editing a suitable hole. This rear-access technique has probably not yet been used by pirates so far, but the technique is about to become much more commonly available and therefore has to be taken into account by designers of new security chips.

FIBs are used by attackers today primarily to simplify manual probing of deep metal and polysilicon lines. A hole is drilled to the signal line of interest, filled with platinum to bring the signal to the surface, where a several micrometer large probing pad or cross is created to allow easy access (Fig. 11). Modern FIB workstations (for example the FIB 200xP from FEI) cost less than half a million US\$ and are available in over hundred organizations. Processing time can be rented from numerous companies all over the world for a few hundred dollars per hour.

Another useful particle beam tool are electron-beam testers (EBT) [14]. These are SEMs with a voltage-contrast function. Typical acceleration voltages and beam currents for the primary electrons are 2.5 kV and 5 nA. The number and energy of secondary electrons are an indication of the local electric field on the chip surface and signal lines can be observed with submicrometer resolution. The signal generated during e-beam testing is essentially the low-pass filtered product of the beam current multiplied with a function of the signal voltage, plus noise. EBTs can measure waveforms with a bandwidth of several gigahertz, but only with periodic signals where stroboscopic techniques and periodic averaging can be used. If we use real-time voltage-contrast mode, where the beam is continuously di-

rected to a single spot and the blurred and noisy stream of secondary electrons is recorded, then the signal bandwidth is limited to a few megahertz [14]. While such a bandwidth might just be sufficient for observing a single signal line in a 3.5 MHz smartcard, it is too low to observe an entire bus with a sample frequency of several megahertz for each line.

EBTs are very convenient attack tools if the clock frequency of the observed processor can be reduced below 100 kHz to allow real-time recording of all bus lines or if the processor can be forced to generate periodic signals by continuously repeating the same transaction during the measurement.

2.2 Non-invasive Attacks

A processor is essentially a set of a few hundred flipflops (registers, latches, and SRAM cells) that define its current state, plus combinatorial logic that calculates from the current state the next state during every clock cycle. Many analog effects in such a system can be used in non-invasive attacks. Some examples are:

- Every transistor and interconnection have a capacitance and resistance that, together with factors such as the temperature and supply voltage, determine the signal propagation delays. Due to production process fluctuations, these values can vary significantly within a single chip and between chips of the same type.
- A flipflop samples its input during a short time interval and compares it with a threshold voltage derived from its power supply voltage. The time of this sampling interval is fixed relative to the clock edge, but can vary between individual flipflops.
- The flipflops can accept the correct new state only after the outputs of the combinatorial logic have stabilized on the prior state.
- During every change in a CMOS gate, both the p- and n-transistors are open for a short time, creating a brief short circuit of the power supply lines [15]. Without a change, the supply current remains extremely small.
- Power supply current is also needed to charge or discharge the load capacitances when an output changes.
- A normal flipflop consists of two inverters and two transmission gates (8 transistors). SRAM cells use only two inverters and two transistors

to ground one of the outputs during a write operation. This saves some space but causes a significant short-circuit during every change of a bit.

There are numerous other effects. During careful security reviews of processor designs it is often necessary to perform detailed analog simulations and tests and it is not sufficient to just study a digital abstraction.

Smartcard processors are particularly vulnerable to non-invasive attacks, because the attacker has full control over the power and clock supply lines. Larger security modules can be equipped with backup batteries, electromagnetic shielding, low-pass filters, and autonomous clock signal generators to reduce many of the risks to which smartcard processors are particularly exposed.

2.2.1 Glitch Attacks

In a glitch attack, we deliberately generate a malfunction that causes one or more flipflops to adopt the wrong state. The aim is usually to replace a single critical machine instruction with an almost arbitrary other one. Glitches can also aim to corrupt data values as they are transferred between registers and memory. Of the many fault-induction attack techniques on smartcards that have been discussed in the recent literature [11, 12, 16, 17, 18], it has been our experience that glitch attacks are the ones most useful in practical attacks.

We are currently aware of three techniques for creating fairly reliable malfunctions that affect only a very small number of machine cycles in smartcard processors: clock signal transients, power supply transients, and external electrical field transients.

Particularly interesting instructions that an attacker might want to replace with glitches are conditional jumps or the test instructions preceding them. They create a window of vulnerability in the processing stages of many security applications that often allows us to bypass sophisticated cryptographic barriers by simply preventing the execution of the code that detects that an authentication attempt was unsuccessful. Instruction glitches can also be used to extend the runtime of loops, for instance in serial port output routines to see more of the memory after the output buffer [12], or also to reduce the runtime of loops, for instance to transform an iterated cipher function into an easy to break single-round variant [11].

Clock-signal glitches are currently the simplest and most practical ones. They temporarily increase the clock frequency for one or more half cycles, such that some flipflops sample their input before the new

state has reached them. Although many manufacturers claim to implement high-frequency detectors in their clock-signal processing logic, these circuits are often only simple-minded filters that do not detect single too short half-cycles. They can be circumvented by carefully selecting the duty cycles of the clock signal during the glitch.

In some designs, a clock-frequency sensor that is perfectly secure under normal operating voltage ignores clock glitches if they coincide with a carefully designed power fluctuation. We have identified clock and power waveform combinations for some widely used processors that reliably increment the program counter by one without altering any other processor state. An arbitrary subsequence of the instructions found in the card can be executed by the attacker this way, which leaves very little opportunity for the program designer to implement effective countermeasures in software alone.

Power fluctuations can shift the threshold voltages of gate inputs and anti-tampering sensors relative to the unchanged potential of connected capacitances, especially if this occurs close to the sampling time of the flipflops. Smartcard chips do not provide much space for large buffer capacitors, and voltage threshold sensors often do not react to very fast transients.

In a potential alternative glitch technique that we have yet to explore fully, we place two metal needles on the card surface, only a few hundred micrometers away from the processor. We then apply spikes of a few hundred volts for less than a microsecond on these needles to generate electrical fields in the silicon substrate of sufficient strength to temporarily shift the threshold voltages of nearby transistors.

2.2.2 Current Analysis

Using a 10–15 Ω resistor in the power supply, we can measure with an analog/digital converter the fluctuations in the current consumed by the card. Preferably, the recording should be made with at least 12-bit resolution and the sampling frequency should be an integer multiple of the card clock frequency.

Drivers on the address and data bus often consist of up to a dozen parallel inverters per bit, each driving a large capacitive load. They cause a significant power-supply short circuit during any transition. Changing a single bus line from 0 to 1 or vice versa can contribute in the order of 0.5–1 mA to the total current at the right time after the clock edge, such that a 12-bit ADC is sufficient to estimate the number of bus bits that change at a time. SRAM write operations often generate the strongest

signals. By averaging the current measurements of many repeated identical transactions, we can even identify smaller signals that are not transmitted over the bus. Signals such as carry bit states are of special interest, because many cryptographic key scheduling algorithms use shift operations that single out individual key bits in the carry flag. Even if the status-bit changes cannot be measured directly, they often cause changes in the instruction sequencer or microcode execution, which then cause a clear change in the power consumption.

The various instructions cause different levels of activity in the instruction decoder and arithmetic units and can often be quite clearly distinguished, such that parts of algorithms can be reconstructed. Various units of the processor have their switching transients at different times relative to the clock edges and can be separated in high-frequency measurements.

3 Countermeasures

3.1 Randomized Clock Signal

Many non-invasive techniques require the attacker to predict the time at which a certain instruction is executed. A strictly deterministic processor that executes the same instruction c clock cycles after each reset—if provided with the same input at every cycle—makes this easy. Predictable processor behaviour also simplifies the use of protocol reaction times as a covert channel.

The obvious countermeasure is to insert random-time delays between any observable reaction and critical operations that might be subject to an attack. If the serial port were the only observable channel, then a few random delay routine calls controlled by a hardware noise source would seem sufficient. However, since attackers can use cross-correlation techniques to determine in real-time from the current fluctuations the currently executed instruction sequence, almost every instruction becomes an observable reaction, and a few localized delays will not suffice.

We therefore strongly recommend introducing timing randomness at the clock-cycle level. A random bit-sequence generator that is operated with the external clock signal should be used to generate an internal clock signal. This will effectively reduce the clock frequency by a factor of four, but most smartcards anyway reduce internally the 3.5 MHz provided for contact cards and the 13 MHz provided for contact-less cards.

Hardware random bit generators (usually the amplified thermal noise of transistors) are not always

good at producing uniform output statistics at high bit rates, therefore their output should be smoothed with an additional simple pseudo-random bit generator.

The probability that n clock cycles have been executed by a card with a randomized clock signal after c clock cycles have been applied can be described as a binomial distribution:

$$p(n, c) = 2^{-c} \left[\binom{c}{2n} \binom{c}{2n+1} \right] \\ \approx \sqrt{\frac{8}{\pi c}} \cdot e^{-\frac{8}{c} \cdot (n - \frac{c}{4})^2} \quad \text{as } c \rightarrow \infty$$

So for instance after we have sent 1000 clock cycles to the smartcard, we can be fairly sure (probability $> 1 - 10^{-9}$) that between 200 and 300 of them have been executed. This distribution can be used to verify that safety margins for timing-critical algorithms—such as the timely delivery of a pay-TV control word—are met with sufficiently high probability.

Only the clock signals of circuitry such as the serial port and timer need to be supplied directly with the external clock signal, all other processor parts can be driven from the randomized clock.

A lack of switching transients during the inactive periods of the random clock could allow the attacker to reconstruct the internal clock signal from the consumed current. It is therefore essential that the processor shows a characteristic current activity even during the delay phases of the random clock. This can be accomplished by driving the bus with random values or by causing the microcode to perform a write access to an unused RAM location while the processor is inactive.

3.2 Randomized Multithreading

To introduce even more non-determinism into the execution of algorithms, it is conceivable to design a multithreaded processor architecture [19] that schedules the processor by hardware between two or more threads of execution randomly at a per-instruction level. Such a processor would have multiple copies of all registers (accumulator, program counter, instruction register, etc.), and the combinatorial logic would be used in a randomly alternating way to progress the execution state of the threads represented by these respective register sets.

The simple 8-bit microcontrollers of smartcards do not feature pipelines and caches and the entire state is defined only by a very small number of registers that can relatively easily be duplicated. The only other necessary addition would be new machine

instructions to fork off the other thread(s) and to synchronize and terminate them. Multithreaded applications could interleave some of the many independent cryptographic operations needed in security protocols. For the remaining time, the auxiliary threads could just perform random encryptions in order to generate an realistic current pattern during the delay periods of the main application.

3.3 Robust Low-frequency Sensor

Bus-observation by e-beam testing becomes much easier when the processor can be clocked with only a few kilohertz, and therefore a low-frequency alarm is commonly found on smartcard processors. However, simple high-pass or low-pass RC elements are not sufficient, because by carefully varying the duty cycle of the clock signal, we can often prevent the activation of such detectors. A good low-frequency sensor must trigger if no clock edge has been seen for longer than some specified time limit (e.g., $0.5 \mu\text{s}$). In this case, the processor must not only be reset immediately, but all bus lines and registers also have to be grounded quickly, as otherwise the values on them would remain visible sufficiently long for a voltage-contrast scan.

Even such carefully designed low-frequency detectors can quite easily be disabled by laser cutting or FIB editing the RC element. To prevent such simple tampering, we suggest that an intrinsic self-test be built into the detector. Any attempt to tamper with the sensor should result in the malfunction of the entire processor. We have designed such a circuit that tests the sensor during a required step in the normal reset sequence. External resets are not directly forwarded to the internal reset lines, but only cause an additional frequency divider to reduce the clock signal. This then activates the low-frequency detector, which then activates the internal reset lines, which finally deactivate the divider. The processor has now passed the sensor test and can start normal operation. The processor is designed such that it will not run after a power up without a proper internal reset. A large number of FIB edits would be necessary to make the processor operational without the frequency sensor being active.

Other sensor defenses against invasive attacks should equally be embedded into the normal operation of the processor, or they will easily be circumvented by merely destroying their signal or power supply connections.

3.4 Destruction of Test Circuitry

Microcontroller production has a yield of typically around 95%, so each chip has to be thoroughly tested

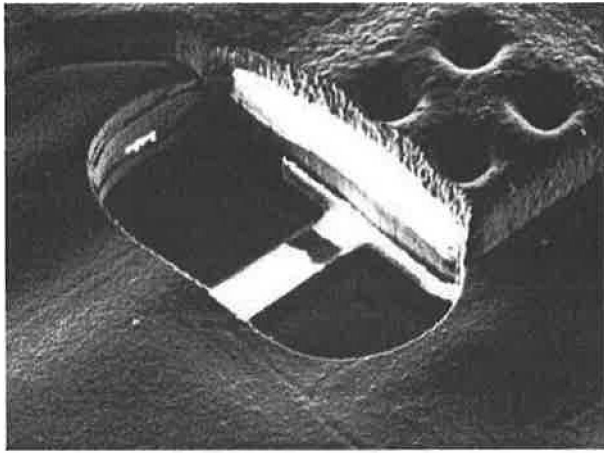


Figure 8: The interrupted white line at the bottom of the cavity in this FIB secondary-electron image is a blown polysilicon fuse next to a test pad (MC68HC05SC2 x processor).

after production. Test engineers—like microprobing attackers—have to get full access to a complex circuit with a small number of probing needles. They add special test circuitry to each chip, which is usually a parallel/serial converter for direct access to many bus and control lines. This test logic is accessible via small probing pads or multiplexed via the normal I/O pads. On normal microcontrollers, the test circuitry remains fully intact after the test. In smartcard processors, it is common practice to blow polysilicon fuses that disable access to these test circuits (Fig. 8). However, attackers have been able to reconnect these with microprobes or FIB editing, and then simply used the test logic to dump the entire memory content.

Therefore, it is essential that any test circuitry is not only slightly disabled but structurally destroyed by the manufacturer. One approach is to place the test interface for chip n onto the area of chip $n + 1$ on the wafer, such that cutting the wafer into dies severs all its parallel connections. A wafer saw usually removes a 80–200 μm wide area that often only contains a few process control transistors. Locating essential parts of the test logic in these cut areas would eliminate any possibility that even substantial FIB edits could reactivate it.

3.5 Restricted Program Counter

Abusing the program counter as an address pattern generator significantly simplifies reading out the entire memory via microprobing or e-beam testing.

Separate watchdog counters that reset the processor if no jump, call, or return instruction is executed

for a number of cycles would either require many transistors or are too easily disabled.

Instead, we recommend simply not providing a program counter that can run over the entire address space. A 16-bit program counter can easily be replaced with the combination of a say 7-bit offset counter O and a 16-bit segment register S , such that the accessed address is $S + O$. Instead of overflowing, the offset counter resets the processor after reaching its maximum value. Every jump, call, or return instruction writes the destination address into S and resets O to zero. The processor will now be completely unable to execute more than 127 bytes of machine code without a jump, and no simple FIB edit will change this. A simple machine-code post-processor must be used by the programmer to insert jumps to the next address wherever unconditional branches are more than 127 bytes apart.

With the program counter now being unavailable, attackers will next try to increase the number of iterations in software loops that read data arrays from memory to get access to all bytes. This can for instance be achieved with a microprobe that performs a glitch attack directly on a bus-line. Programmers who want to use 16-bit counters in loops should keep this in mind.

3.6 Top-layer Sensor Meshes

Additional metallization layers that form a sensor mesh above the actual circuit and that do not carry any critical signals remain one of the more effective annoyances to microprobing attackers. They are found in a few smartcard CPUs such as the ST16SF48A or in some battery-buffered SRAM security processors such as the DS5002FPM and DS1954.

A sensor mesh in which all paths are continuously monitored for interruptions and short-circuits while power is available prevents laser cutter or selective etching access to the bus lines. Mesh alarms should immediately trigger a countermeasure such as zeroizing the non-volatile memory. In addition, such meshes make the preparation of lower layers more difficult, because since the etch progresses unevenly through them, their pattern remains visible in the layers below and therefore they complicate automatic layout reconstruction. Finally, a mesh on top of a polished oxide layer hides lower layers, which makes navigation on the chip surface for probing and FIB editing more tedious.

The implementations of sensor meshes in fielded products however show a number of quite surprising design flaws that significantly reduce the protection (Fig. 9 and 10). The most significant flaw is

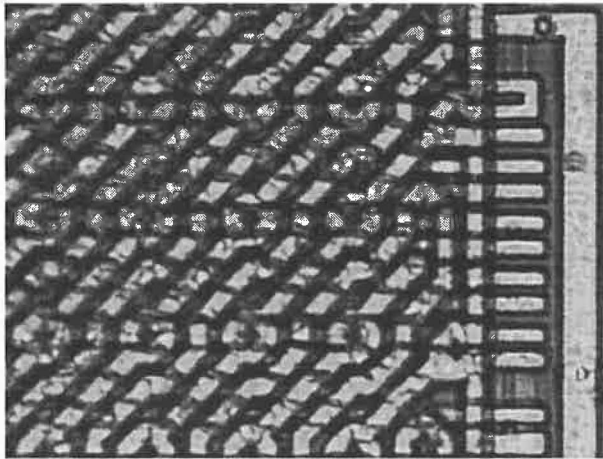


Figure 9: Escape route for imprisoned crypto bits: The ST16SF48A designers generously added this redundant extension of the data bus several micrometers beyond the protected mesh area, providing easy probing access.

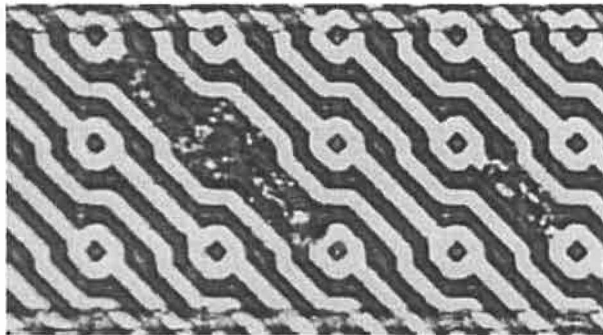


Figure 10: Every second line is connected to VCC or GND at one end and open at the other. Not all are used to supply lower layers and therefore some can safely be opened with a laser for probing access to the bus lines below.

that a mesh breach will only set a flag in a status register and that zeroization of the memory is left completely to the application software. We noted in Section 2.1.4 that a common read-out technique involves severely disabling the instruction decoder, therefore software checks for invasive attacks are of little use.

A well-designed mesh can make attacks by manual microprobing alone rather difficult, and more sophisticated FIB editing procedures will be required to bypass it. Several techniques can be applied here. The resolution of FIB drilling is much smaller than the mesh line spacings, therefore it is no problem to establish contact through three or more metal layers and make deeply buried signals accessible for micro-

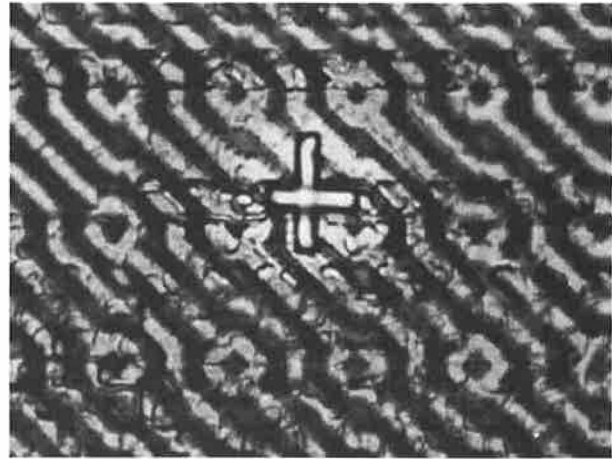


Figure 11: A FIB was used here to drill a fine hole to a bus line through the gap between two sensor mesh lines, refill it with metal, and place a metal cross on top for easy microprobing access.

probing via a platinum or tungsten pad on top of the passivation layer (Fig. 11). Alternatively, it is also possible to etch a larger window into the mesh and then reconnect the loose ends with FIB metal deposits around it.

4 Conclusion

We have presented a basis for understanding the mechanisms that make microcontrollers particularly easy to penetrate. With the restricted program counter, the randomized clock signal, and the tamper-resistant low-frequency sensor, we have shown some selected examples of low-cost countermeasures that we consider to be quite effective against a range of attacks.

There are of course numerous other more obvious countermeasures against some of the commonly used attack techniques which we cannot cover in detail in this overview. Examples are current regulators and noisy loads against current analysis attacks and loosely coupled PLLs and edge barriers against clock glitch attacks. A combination of these together with e-field sensors and randomized clocks or perhaps even multithreading hardware in new processor designs will hopefully make high-speed non-invasive attacks considerably less likely to succeed. Other countermeasures in fielded processors such as light and depassivation sensors have turned out to be of little use as they can be easily bypassed.

We currently see no really effective short-term protection against carefully planned invasive tampering involving focused ion-beam tools. Zeroization mechanisms for erasing secrets when tampering

is detected require a continuous power supply that the credit-card form factor does not allow. The attacker can thus safely disable the zeroization mechanism before powering up the processor. Zeroization remains a highly effective tampering protection for larger security modules that can afford to store secrets in battery-backed SRAM (e.g., DS1954 or IBM 4758), but this is not yet feasible for the smartcard package.

5 Acknowledgements

The authors would like to thank Ross Anderson, Simon Moore, Steven Weingart, Matthias Brunner, Gareth Evans and others for useful and highly interesting discussions.

References

- [1] FIPS PUB 140-1: Security Requirements for Cryptographic Modules. National Institute of Standards and Technology, U.S. Department of Commerce, 11 January 1994.
- [2] F. Beck: *Integrated Circuit Failure Analysis – A Guide to Preparation Techniques*. John Wiley & Sons, 1998.
- [3] T.W. Lee, S.V. Pabbisetty (eds.): *Microelectronic Failure Analysis, Desk Reference*. 3rd edition, ASM International, Ohio, 1993, ISBN 0-87170-479-X.
- [4] N.H.E. Weste, K. Eshraghian: *Principles of CMOS VLSI Design*. Addison-Wesley, 1993.
- [5] S.-M. Kang, Y. Leblebici: *CMOS Digital Integrated Circuits: Analysis and Design*. McGraw-Hill, 1996.
- [6] J. Carter: *Microprocessor Architecture and Microprogramming – A State-Machine Approach*. Prentice-Hall, 1996.
- [7] S.M. Sze: *Semiconductor Devices – Physics and Technology*. John Wiley & Sons, 1985.
- [8] T.R. Corle, G.S. Kino: *Confocal Scanning Optical Microscopy and Related Imaging Systems*. Academic Press, 1996.
- [9] S. Blythe, et al.: Layout Reconstruction of Complex Silicon Chips. *IEEE Journal of Solid-State Circuits*, 28(2):138–145, February 1993.
- [10] D.P. Maher: Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective. In R. Hirschfeld (ed.): *Financial Cryptography, FC '97*, Proceedings, LNCS 1318, pp. 109–121, Springer-Verlag, 1997.
- [11] R.J. Anderson, M.G. Kuhn: Low Cost Attacks on Tamper Resistant Devices. In M. Lomas, et al. (eds.), *Security Protocols, 5th International Workshop*, LNCS 1361, pp. 125–136, Springer-Verlag, 1997.
- [12] R.J. Anderson, M.G. Kuhn: Tamper Resistance — a Cautionary Note. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pp. 1–11, Oakland, California, 18–21 November 1996.
- [13] J.H. Daniel, D.F. Moore, J.F. Walker: Focused Ion Beams for Microfabrication. *Engineering Science and Education Journal*, pp. 53–56, April 1998.
- [14] H. P. Feuerbaum: Electron Beam Testing: Methods and Applications. *Scanning*, 5(1):14–24, 1982.
- [15] H.J.M. Veendrick: Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits. *IEEE Journal of Solid-State Circuits*, 19(4):468–473, August 1984.
- [16] D. Boneh, R.A. DeMillo, R.J. Lipton: On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology – EUROCRYPT '97*, LNCS 1233, pp. 37–51, Springer-Verlag, 1997.
- [17] F. Bao, et al.: Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. In M. Lomas, et al. (eds.), *Security Protocols, 5th International Workshop*, LNCS 1361, pp. 115–124, Springer-Verlag, 1997.
- [18] M. Joye, J.-J. Quisquater, F. Bao, R. H. Deng: RSA-type Signatures in the Presence of Transient Faults. In *Cryptography and Coding*, LNCS 1355, pp. 155–160, Springer-Verlag, 1997.
- [19] S.W. Moore: *Multithreaded Processor Design*. Kluwer Academic Publishers, 1996.

Which Security Policy for Multiapplication Smart Cards?

Pierre Girard

Cryptography and Security R&D

GEMPLUS

Parc d'activité de Gémenos - B.P. 100

13881 Gémenos CEDEX - France

Pierre.Girard@gemplus.com

Abstract

In this paper, we aim to clarify some issues regarding the deployment context of multiapplicative smart cards. We especially deal with the trust relationships between the involved parties and the resulting constraints from a security point of view.

We highlight a new security threat in a multiapplicative context and propose a new multilevel security model which allows to control precisely the information flows inside the card, and to detect illegal data sharing.

Finally we illustrate all the proposed concepts on an multiapplicative example involving three applications.

1 Introduction

Multiapplication smart cards are getting more and more attractive for numerous good reasons. Users are willing to reduce the number of cards in their wallets, issuers want to decrease the time-to-market, the development, infrastructure and deployment costs or to update/add applications after card issuance. In addition multiapplication smart cards allow commercial synergies between partners and can lead to new business opportunities. A credit card with an electronic purse and a frequent flyer application is a classical example of a multiapplication smart card.

A few operating systems have been proposed to manage multiapplicative smart cards, namely Java

Card¹, Multos² and more recently Smart Card for Windows³. In this paper we will focus on Java Card and exhibit examples for this platform, but all the results can apply to any multiapplicative platform.

Security is always a big concern for smart cards, but the issue is getting more intense with multiapplicative platforms and post issuance code downloading. Of course, Java Card security or protection against aggressive applets have been discussed extensively, but we still lack a global security policy for multiapplicative smart cards. We will show that this kind of policy must be more than the simple concatenation of the security policy of all applications. Until now, this topic hasn't been appropriately investigated, probably because numerous issues concerning how those cards will be used remain unclear: which parties will cooperate? how? which of them will drive the scheme? etc.

In this paper we first aim at clarifying those issues. Next, we show that there is a need for a card-wide security policy, mainly because of the existence of information sharing mechanisms between applets. Then, we propose a new security policy to deal with this need and show an example of how it could be used. We finish with limitations, potential extensions and related work.

2 The multiapplication platform

On a multiapplication platform, we usually find an operating system managing the card resources (like I/O, memory, random number generator, crypto en-

¹See <http://java.sun.com/products/javacard>.

²See <http://www.multos.com>.

³See <http://www.microsoft.com/smartcard>.

gine. . .) and some applications (possibly loaded after the card issuance) from various sources using the OS services. In addition, the OS ensures application segregation and provides mechanisms to allow controlled data sharing between applications.

2.1 Which participants?

Unlike single-application smart cards, multiapplication smart cards involve many participants. Of course, there are still an issuer and an end user, but additional third parties which can interact directly or not with the card are also involved.

To separate precisely the actions of the participants we have defined two roles in addition to the usual issuer which is played by a unique authority: the application provider and the card operator.

The application provider designs an application for the targeted card operating system and negotiates with the issuer for downloading its application inside the card (before or after the card issuance). It's the owner of the applet and applet's data. Of course, the issuer itself will place some applications inside its card, and will play the application provider's role.

The card operator is an entity which can interact with the card either to use an application or to perform administrative tasks. An administrative task can be anything from auditing the card or updating a key to downloading a new application. The interaction between the operator and the card can be direct (e.g. if the card is inserted in an ATM) or remote (e.g. through the Internet or a cellular phone network). It is likely that the application providers and the issuer will play the operator's role as well. Conversely, an operator will probably be an application provider, even if it could be delegated to interact with an application of another provider. An applet can be designed to work with one operator (likely its owner) or more. In this case its behaviour could differ according to the operator.

Coming back to the example of a credit card with a frequent flyer application, we can assume that the issuer will be a bank, which will also be the application provider and the operator of the credit applet. An air travel company will be the provider and the operator of the loyalty applet. Some other companies can join the loyalty program and become

operators for the loyalty applet.

2.2 How is it operated?

The first and the simplest way to operate such a multiapplication platform is to keep it entirely under the control of the issuer. It will be the only authority responsible for the card and its integrity. All application providers must negotiate and agree with the issuer conditions and guidelines for downloading their applications on the card. The issuer is granted all possible rights in the card and will enforce its policy during all the card life, as well as inspect applications carefully before downloading them. In the rest of the paper we suppose this mode of operation as it seems to be the most likely in the forthcoming years. However, other ones like the two following could exist.

One can imagine a second model where the end user buys a blank card from a manufacturer of its choice and plays the issuer's role. Afterwards, it will accept or buy from providers some applications for its card. The whole card will be (or should be) under its control. We do not consider this scenario as it is unclear if providers will accept to download their applications into cards for which they have no or few behaviour guarantees. Probably, this type of scheme will coexist with the first one, but for non security critical applications or for applications totally operated by the end user. For example, a manufacturer of smart locks using smart cards, instead of keys, could propose to download its application on users cards. The users could decide to buy an *ad hoc* card or use an existing one to put the application driving the smart lock.

A third type of scheme adds another role: certification authorities. Those authorities will audit issuers and their cards and will provide a certification which will guarantee that an issuer respects a given policy. Based on this policy a provider as well as an end user will be able to decide if the issuer's policy complies with their requirements. An example of certification could be a "privacy awareness" label for issuers which respect the privacy rights of the users and do not leak private data outside the card. This is a refinement of our first scheme and we won't focus on it in the following.

2.3 Which security policy?

At first glance, it is easy to see that there are, at least, two separate security problems: the platform level security and the application level security.

The first one (platform level security) concerns applications segregation (this can be viewed as the classic OS security) as well as the quality of security services offered by the platform (e.g. correctness of the Java virtual machine including the verifier, tamper resistance, cryptographic algorithms and post-issuance loading mechanism). This part is under the issuer's responsibility.

The second one (application security), is under the provider's responsibility, but relies necessarily on the platform security. Moreover, the application should assume that the OS won't be aggressive and will act as it is supposed to. Conversely, the OS doesn't make any assumption about the application and should still work and protect the other applications even if an aggressive or insecure piece of code is loaded. However, one should note that even if this is technically perfectly acceptable, and if an insecure application can't threaten the platform or other application, the end users or potential customers could get confused by a break-in of an application and mixed up the platform security and the application security. To avoid this potential damage to its brand image, an issuer could enforce a minimum security level for the applications loaded on its card by reviewing them or operating a scheme including some certification authorities.

Apart from these two obvious security aspects, a third one must be addressed. All the difficulties arise from data sharing inside a card. Actually, most of multiapplication smart cards, in order to build cooperative schemes and optimize memory usage, allow data and service sharing (i.e. objects sharing) between applications. And beyond this point there is a need for a card-wide security policy concerning all the applications. A small example should make this point clearer. When an application provider *A* decides to share (or more probably to sell) some data with an application provider *B*, he will ask for guarantees that *B* won't be able to resell those data or make them available world-wide.

To address these problems two kinds of security policies can be introduced: a discretionary one and a mandatory one. The applications will be in charge

of defining their own discretionary security policy which could be enforced by the OS. For an example, in a Java card, an applet can decide to share some of its objects with a selective list of other applets. On this access control list basis, the OS will allow or deny access to the shared object by other applets. If we just consider a discretionary policy, nothing prevents an application *B* which could legally access an object of *A* (*B* has been granted from *A* to read the data) from copying the information into another object shared with *C*. In this case the information is transferred from *A* to *C* even if *C* is not granted access the information from *A*.

A mandatory security policy is necessary to solve the problem of re-sharing shared objects as pointed above. Actually none of the existing OS enforce such a policy.

In this paper, we will discuss this last point and propose a security model, compliant with the requirements of safe data sharing and able to control the information flows inside the card.

2.4 Which threat model?

In the following we consider the threat of an insecure or malicious applet gaining legally some access to sensitive data (by a sharing mechanism of the OS) but resharing them illegally with a third party.

The threat could be a commercial concern (as in the previous example) or privacy concern. This later case is especially important when dealing with health-care cards containing medical records or with loyalty cards containing marketing profiles.

3 A new security model

The security policy enforced by the OS should model the information flows between the applications which, themselves, reflect the trust relationships between the participants of the applicative scheme. In this section we will start by studying the trust relationships, then the security model, and finally consider some implementation issues.

3.1 Trust relationships

In the basic situation the only trust relationships are from everyone to the issuer, as there is no way for a participant to distrust the issuer. The platform OS is completely under the issuer's control and is potentially able to read, write, create or modify everything on it including the applications and their keys.

In addition, some participants can trust other ones: sometimes because it is in fact the same entity which plays more than one role and sometimes because there is an agreement or a contract between the two.

It should be noticed that the trust relationship is neither symmetric nor transitive: an entity wouldn't like to trust someone only because one of its partners trusts it. This situation is not likely to change as co-operation in industry becomes something more complex and subject to daily change (one could speak about co-competition between companies as well as between divisions inside a company).

Figure 1 shows an example of trust relationships with four applications providers and three operators. The issuer trusts no-one except OP1 and AP1 which are two roles played by the issuer itself. We can see that AP1 trusts AP2 and AP2 trusts AP3 which doesn't mean that AP1 trusts AP3.

We can also note that AP4 doesn't operate its applet itself but relies on, and thus trusts, OP3 to do so.

Now, it should be clear that a mandatory security policy must be enforced in a multiapplicative scheme which will allow or deny data flows between applications given the trust relationships. If an entity *A* trusts an entity *B*, this means that some information could flow from *A* to *B*. We have chosen a classic multilevel security policy using a set of security level modelling the trust relationships.

3.2 The multilevel policy

The multilevel security policy, first modelled in [2], uses a set of security levels with a complete lattice structure. A security level is associated to each subject (an entity manipulating information) and each object (a piece of information but not an object in object-oriented programming sense) of the system.

The lattice structure implies an order relation on the set, and hence that the relation is transitive.

The security rule states that a read (resp. write) access by a subject to an object is granted if and only if the level of the subject is greater (resp. lower) or equal to the object's level. The classical example of a multilevel security policy is the document classification inside a military organisation. In this case the set of security level is: {*Unclassified*, *Confidential*, *Secret*, *Top secret*} with a usual order between the security levels.

We will now consider how to map each entity and information in a multiapplicative scheme to a security level. First of all we will create one level associated to the issuer and one associated to each application provider. If an operator and an application provider trust each other they will be represented by only one level.

To establish a multilevel policy for a card without data sharing, one can choose a (non flat) lattice of security levels with one level for each application provider and an additional public level to complete the lattice. Figure 2 indicates the security level lattice corresponding to the example of figure 1, part (a), where an arrow represents an order relation between two levels. In this case, the only legal information flow is from a public level to the issuer through any role, but communication between providers or operators is strictly prohibited except from AP4 to AP3 because AP4 trusts OP3 and because OP3 and AP3 have been merged.

To allow data sharing between entities, one cannot simply allow a new order relation between two roles as we cannot accept the transitivity effect of the order relation. As an example, if AP1 from our last example wants to share some data with AP2 and AP2 wants to share data with AP3, the security lattice of figure 2, part (b) is clearly inadequate. We recall that the trust relationship is not transitive and so AP1 does not want to share data with AP3 which is possible with this solution through AP2.

To solve this problem we propose that AP1 and AP2 agree on a data subset they want to share. Those data will be classified with a new level called AP1+AP2 placed in the security level lattice shown on figure 3. The same agreement will take place between AP2 and AP3 resulting in another level AP2+AP3.

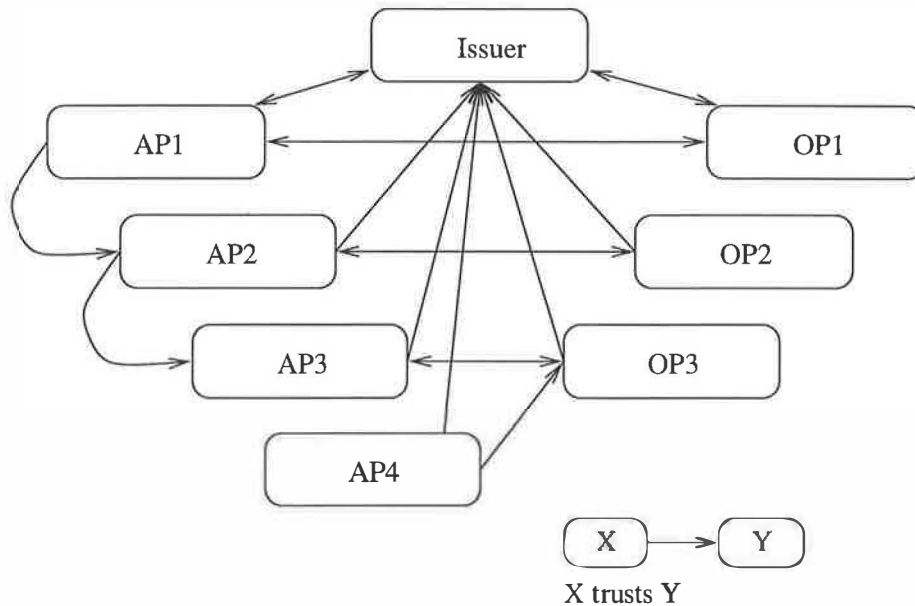


Figure 1: Example of a trust relationship

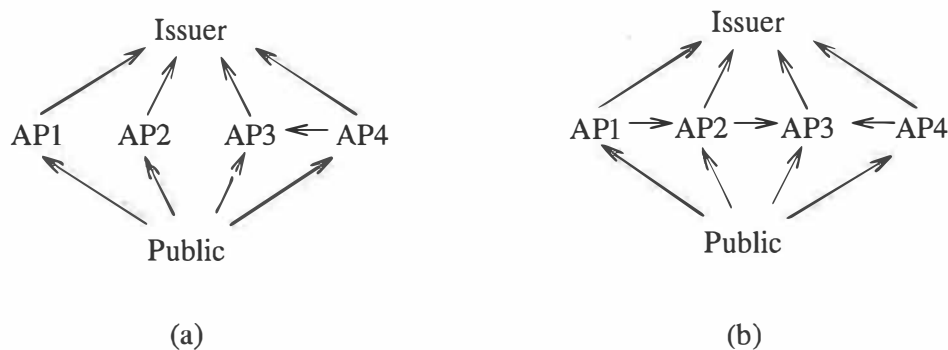


Figure 2: Example of security level lattice without sharing (a) and with an unacceptable sharing (b).

This technique can be refined if there is a need for sharing a subset of AP1+AP2 with AP3. We will create a new level called AP1+AP2+AP3 greater than the public level but lower than AP1+AP2 and AP2+AP3.

3.3 Implementations issues

To enforce the latter policy within a card (we consider here a Java Card), a lot of implementation issues should be considered. First of all, we should decide which data will be classified, and with which granularity. We should also consider the implementation of security mechanisms which will enforce the policy.

The classification of objects in object oriented language is a complex problem (see [5] for a recent discussion on this subject), however, we can chose a simple strategy by assigning a security level to each object instance. In a given applet, the objects will be labelled with their provider's level except if an object is shared, in which case, we will choose the level related to shared data. The authorized information flows in an applet will be from lower labelled objects to higher ones.

Enforcing the security policy could be done dynamically by a reference monitor (part of the card OS) which will be called each time an object reference is used by the virtual machine or statically by checking the correctness of the information flows in an applet. The first solution would be too costly in

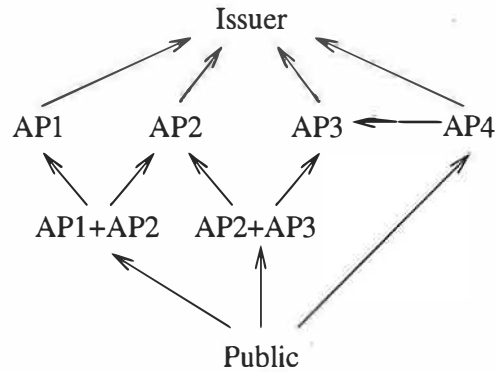


Figure 3: Example of security level lattice with acceptable sharing

memory and execution time which are both critical in a smart card as we should tag each object with its level and check the validity of each read/write operation.

The second solution has been studied for a long time (Denning pioneered this domain [6] and was followed by [8, 1, 11, 4]) and could be integrated to existing static verifier of Java bytecode. Using security level set with a lattice structure is a key point to guarantee that the static analysis will terminate.

Practically, this means that an applet provider will deliver its code to the issuer along with the security level of all the objects contained in it. The issuer will verify that the code and the declared levels of the objects comply with the other applets and their objects security levels.

4 A real world example

We present in this section the example of a multi-applicative healthcare card. This card is issued by a health insurance with an applet. This applet contains some administrative data including the social security number of the card holder.

In addition, the user has joined the loyalty program of a drugstores chain and downloaded the corresponding applet. The drugstore applet can use the social security number and contains some administrative data and, possibly, a few medical records (e.g. medication allergy). It also contains a loyalty part which maintains a loyalty points counter.

The third applet on the user's card is a loyalty applet of a sport centers chain which has an agreement with the drugstores chain and shares the loyalty point counter with the drugstore applet.

Figure 4 summarizes the information flow between the applets.

The drugstore applet can read the social security number, but is not allowed to give it to other applets. The security threat is the following: the drugstore applet can copy the social security number from the healthcare applet to the loyalty point counter and broadcast it to applets allowed to read the counter.

To face this threat, we propose a mandatory security policy using the security level lattice of figure 5. All the objects of the healthcare applet will be labelled with *HC* except the social security number labelled with *SSN*. The objects of the drugstore applet are labelled with the *DS* level except the loyalty point counter which is labelled with *PTS*. Finally, all the objects of the sport centers chain applet are labelled with *SC*.

This way, if the drugstore applet copies the social security number in one of its objects (labelled *DS*) and later in the loyalty point counter (labelled *PTS*), an illegal information flow will be detected as *PTS* is lower than *DS* and information can only flow from a lower level to a greater one.

To be more complete, we should deal with the external world: the drugstore applet can exchange some data with an operator. The external software used by the operator should also be checked to verify that it doesn't content a covert channel transferring infor-

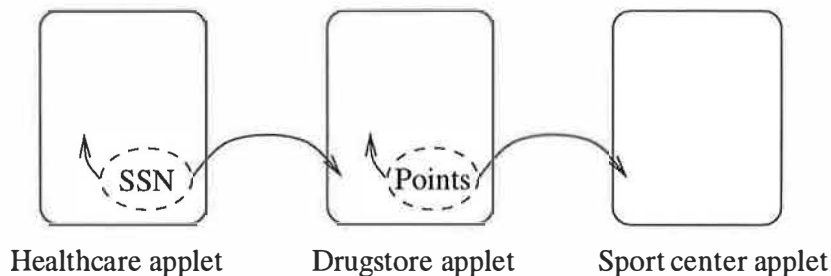


Figure 4: Example of three applets sharing data

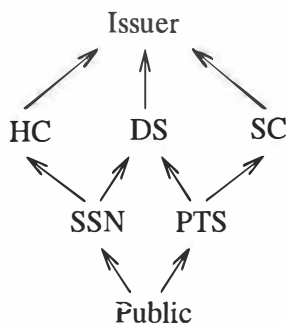


Figure 5: Lattice of security levels for figure 4 applications

mation from the *DS* level to the *PTS* level.

As this software can be modified, the issuer can also ask the drugstore applet provider to encrypt the social security number before sending it outside the card.

5 Limitations and related work

The limitations of our work clearly come from the communications with the external world. We are currently improving this point of the security model. We are also in the process of implementing static verifiers of applets as well as including declassification processes (e.g. when the data flow through a ciphering filter) in the security model.

Some authors have already dealt with non-transitivity constraints in different contexts [10, 3], but we are not aware of a multilevel security policy applied to a smart card and its environment. A lot of papers dealing with classic Java Card security are available. We refer the reader to recent publications like [9] or [7] for a complete bibliography.

6 Conclusion

In this paper, we clarify some issues around the operating scheme of multiapplicative smart cards and highlight some new security threats.

The proposed multilevel security model allows us to control precisely the information flows inside the card, and detect illegal data sharings.

In a next step, analysing tools should be developed and provided to issuers which will be able to audit applets proposed by third parties for their cards.

References

- [1] Jean-Pierre Banâtre, Ciarán Bryce, and Daniel Le Mtayer. Compile-time detection of information flow in sequential programs. In Dieter Gollmann, editor, *Proceedings of ESORICS*, number 875 in LNCS, pages 55–73. Springer, November 1994.
- [2] D. Bell and L. Lapadula. Secure computer systems : Unified exposition and MULTICS in-

terpretation. Technical report ESD-TR-75-306, MITRE Corporation, 1975.

- [3] Pierre Bieber. Formal techniques for an ITSEC-E4 secure gateway. In *Proceedings of the Twelfth Annual Computer Security Applications Conference*, December 1996.
- [4] J. Cazin, P. Girard, C. O'Halloran, and C. Sennett. Formal validation of software for secure systems. In *Formal Methods, Modelling and Simulation for System Engineering*, Saint-Quentin-en-Yvelines, February 1995.
- [5] Frdric Cuppens and Alban Gabillon. Rules for designing multilevel object-oriented databases. In Jean-Jacques Quisquater and *al.*, editors, *Proceedings of ESORICS*, number 1485 in LNCS, pages 159–174. Springer, September 1998.
- [6] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communication of the ACM*, 20(7):504–512, July 1977.
- [7] Jean-Louis Lanet and Antoine Requet. Formal proof of smart card applets correctness. In Jean-Jacques Quisquater and *al.*, editors, *PreProceedings of CARDIS*, Louvain-la-Neuve, September 1998.
- [8] Maasaki Mizuno and David Schmidt. A security flow control algorithm and its denotational semantics correctness proof. *Formal Aspects of Computing*, 4:727–754, 1992.
- [9] Joachim Posegga and Harald Vogt. Byte code verification for java smart cards based on model checking. In Jean-Jacques Quisquater and *al.*, editors, *Proceedings of ESORICS*, number 1485 in LNCS, pages 175–190. Springer, September 1998.
- [10] John Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report CSL-92-02, SRI, December 1992.
- [11] Dennis Volpano and Cynthia Irvine. Secure flow typing. *Computers and Security*, 16(2):137–144, 1997.

Efficient Block Ciphers for Smartcards

Joan Daemen

Proton World Int'l

Zweefvliegtuigstraat 10

B-1130 Brussel, Belgium

daemen.j@protonworld.com

Vincent Rijmen*

K.U.Leuven, Dept. ESAT

Kard. Mercierlaan 94

B-3001 Heverlee, Belgium

vincent.rijmen@esat.kuleuven.ac.be

April 1, 1999

Abstract

We present a family of block ciphers that can be implemented very efficiently on cheap Smartcard processors. The ciphers use a very small amount of RAM and a reasonable amount of ROM. Both cipher execution and key setup/key change are very fast. The ciphers resist theoretical and practical cryptanalytic attacks and in their design timing and power analysis attacks have been taken into account.

1 Introduction

In many applications Smartcards are used as portable secure devices. For their security the applications make use of MAC generation/verification and encryption/decryption using a block cipher. We present a family of block ciphers that are suited for this purpose. Additionally, all these ciphers can be used as efficient one-way function and the variants with block size of 196 bits or higher are efficient compression function to form an iterated cryptographic hash function. The family is named after its first member that was

designed and published: Square [1].

Currently, the Square family consists of three ciphers: Square, with a block length and a key length of 128 bits; BKSQ with a block length of 96 bits and a variable key length (96, 144 or 192 bits); and Rijndael with a variable block length and key length (both can be independently specified at 128, 192 or 256 bits). The three ciphers are designed to be secure against all known cryptographic attacks. For a treatment of cryptographic security and the design rationale we refer to [1, 2, 3]. This paper treats implementation aspects and in particular those specific for Smartcards.

In Section 2 we present the common cipher structure of the Square family. Section 3 discusses the implementation of the ciphers on typical Smartcard processors. Section 4 treats the features of the presented ciphers to thwart attacks that exploit typical weaknesses of cipher implementations on Smartcards. Section 5 lists concrete performance figures.

*F.W.O. Postdoctoral researcher, sponsored by the Fund for Scientific Research - Flanders (Belgium).

2 Cipher structure

A Square cipher is an iterated block cipher: it consists of the repeated application of a *round transformation* that is parameterized by a round key. The round keys are derived from the cipher key by means of a key schedule. The block length is indicated by n , the cipher key length by m and the number of rounds by r .

2.1 The round transformation

The round transformation is composed of four invertible uniform transformations, called *steps*. These steps can be described most easily by thinking of the input as a rectangular byte array. The dimensions of this byte array vary for the different members of the family, and depend on the block size. The four steps are described as follows (cf. Figure 1).

The diffusion step: Every byte is replaced by a linear combination of the bytes within the same column. The bytes are considered as elements in the field $GF(2^8)$.

The dispersion step: A permutation of the bytes over different columns. This is done by shifting the rows of the byte array over different amounts, or by a transposition of the byte array (for Square).

The nonlinear step: A substitution of the bytes by means of a nonlinear lookup table.

The round key addition: The bytes are EXORed with an n -bit round key.

The choice for the operations in the different steps has been influenced by our wish to make the cipher efficiently implementable on Smartcards. The key addition, the dispersion and the nonlinear step all can be implemented using operations on individual bytes, the natural “unit” on an 8-bit processor. In the diffusion step, inter-byte diffusion has to be realised. On a 32-bit processor, this can

be done by using operations like 32-bit rotations, multiplications, . . . , but the use of these operations complicates Smartcard implementations. In the Square family, the diffusion step can be described as a matrix multiplication (cf. Section 3). The coefficients of the multiplication matrix have been selected carefully to provide diffusion that is optimal in a very definite, mathematical sense, while at the same time allowing very efficient implementation on standard Smartcard processors.

2.2 The key schedule

The round keys have length n and $r + 1$ round keys are required: one for every round and a final key addition. The key schedule can be thought to occur in two phases.

Generation of the expanded key: The expanded key is initialized by taking the m -bit cipher key. It is expanded by iteratively attaching m -bit blocks that are computed from the previously attached block by means of an LFSR-like computation. This is repeated until the expanded key has length $n(r + 1)$.

Extraction of round keys: Round key i is taken from the expanded key by taking the i -th n -bit block.

The LFSR computations in the key expansion ensure that any pair of different cipher keys result in a pair of expanded keys with a large Hamming difference. The addition of round constants removes symmetry between the rounds. This is necessary in order to provide resistance against related-key attacks and attacks where the cipher key is known, e.g., if the cipher is used as the compression function of a hash function.

3 Specific Smartcard implementation aspects

In this section we discuss the implementation of the cipher on 8-bit processors with a

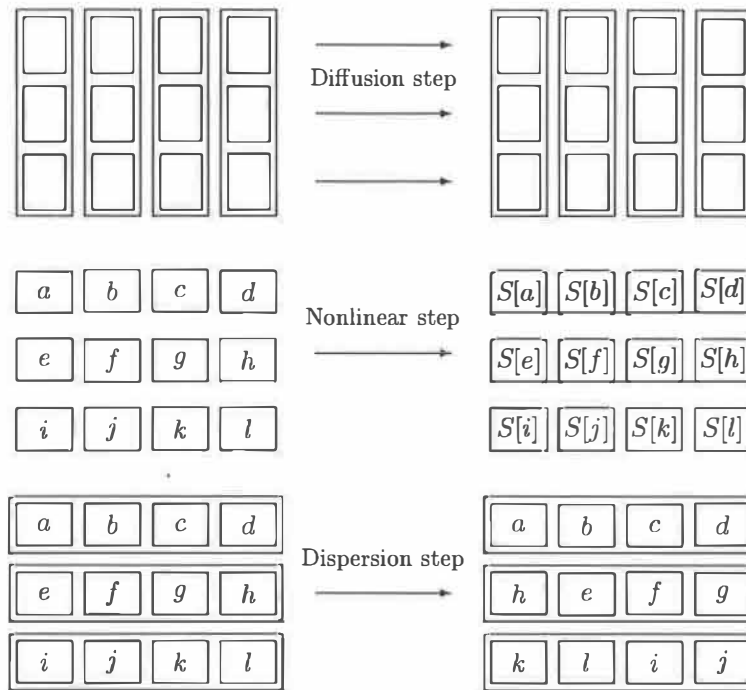


Figure 1: Graphical illustration of some basic operations of the Square ciphers.

limited amount of RAM and ROM available, i.e., typical Smartcard processors.

3.1 The round transformation

The round transformation can be implemented by serially performing the different steps.

The nonlinear step consists of a table lookup operation that is the same for all input bytes. The 256-byte lookup table is hard-coded in the cipher program and the table lookup can be implemented with a simple load accumulator instruction in indexed mode. The round key addition is implemented with the EXOR instruction. The byte dispersion step does not take dedicated instructions but is embodied in the way input bytes are loaded and stored in the preceding/succeeding steps. These three steps can be implemented in the following way: the byte is loaded into the index register, an indexed load accumulator instruction is performed, the round key byte is EXORed and the accumulator is stored to the (hard coded)

location.

Implementing the diffusion step is less straightforward. It takes the computation of additions and multiplication in the field $GF(2^8)$. Addition over this field corresponds to the readily available EXOR instruction. The multiplication factors are the elements of $GF(2^8)$ represented by byte values 1, 2 and 3. The multiplication by these factors can be done as follows:

- 1 is the identity in $GF(2^8)$ and multiplication by it does not require any computation at all.
- Multiplication by 2 in the finite field could be implemented as a left shift, followed by a reduction. However, the execution time and/or the power consumption pattern of a reduction depend on the value of the operand. If the MSB of the operand is 1, the reduction takes place, if 0, the reduction can be skipped. This can be done in constant time by executing dummy instructions (e.g., NOP) in the case the reduction is skipped. However, this gives rise to two different se-

quences of operations. The operation can be implemented with a fixed series of instructions by implementing the multiplication by 2 as a table lookup with a dedicated lookup table $2mult[\cdot]$, that is defined as

$$2mult[a] = 2 \cdot a.$$

The fact that the execution time is independent of the argument makes this table lookup implementation timing attack resistant. We explain in Section 4 how it can be protected against power analysis.

- Multiplication by 3 can be obtained by performing multiplication with 2 and adding (EXORing) the argument itself: $3 \cdot a = (2 \oplus 1) \cdot a = 2mult[a] \oplus a$.

In Rijndael and Square the columns consist of 4 bytes each and the diffusion step applied to a column can be described by a matrix multiplication, that is given by:

$$\begin{bmatrix} out_0 \\ out_1 \\ out_2 \\ out_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} in_0 \\ in_1 \\ in_2 \\ in_3 \end{bmatrix}.$$

This can be efficiently implemented as:

$$\begin{aligned} p &= in_0 \oplus in_1 \oplus in_2 \oplus in_3; \\ out_0 &= 2mult[in_0 \oplus in_1]; out_0 \oplus = p \oplus in_0; \\ out_1 &= 2mult[in_1 \oplus in_2]; out_1 \oplus = p \oplus in_1; \\ out_2 &= 2mult[in_2 \oplus in_3]; out_2 \oplus = p \oplus in_2; \\ out_3 &= 2mult[in_3 \oplus in_0]; out_3 \oplus = p \oplus in_3; \end{aligned}$$

This implementation takes only 15 EXORS and 4 table lookups per column. It requires temporary storage for 2 bytes: the variables p and in_0 (if the output buffer is equal to the input buffer).

In BKSQ the columns consist of 3 bytes and the diffusion step is given by

$$\begin{bmatrix} out_0 \\ out_1 \\ out_2 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & 2 \\ 2 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} in_0 \\ in_1 \\ in_2 \end{bmatrix}.$$

This can be implemented with only 5 EXORS and a single table lookup per column and only one byte of temporary storage is needed.

3.2 The inverse round transformation

The Square ciphers do not have the Feistel structure, like e.g. the DES. Whereas for Feistel ciphers the operation of the cipher can be inverted by simply reordering the round keys, this is not possible for the Square ciphers. Therefore, if the inverse operation of the cipher has to be implemented, it is necessary to implement the inverse round operation separately.

The inverse of the round key addition is the same as the round key addition: EXOR with the round key. The inverse of the non-linear step is implemented like the linear step, but uses a different lookup table. The byte dispersion step is again embodied in the way input bytes are loaded and stored in the other steps.

The inverse of the diffusion step consists of a multiplication with the inverse matrix. For Rijndael and Square, the inverse of the diffusion step is given by:

$$\begin{bmatrix} out_0 \\ out_1 \\ out_2 \\ out_3 \end{bmatrix} = \begin{bmatrix} E & 9 & D & B \\ B & E & 9 & D \\ D & B & E & 9 \\ 9 & D & B & E \end{bmatrix} \cdot \begin{bmatrix} in_0 \\ in_1 \\ in_2 \\ in_3 \end{bmatrix}.$$

Here B, D and E denote hexadecimal values. It is easy to see that this multiplication will require more operations than the original diffusion step, namely 21 EXORS and 8 table lookups using only the previously defined table $2mult[\cdot]$. If additional tables are used, the performance loss can be reduced. The storage requirements do not increase.

Note that most applications do not require the inverse operation of the cipher to be executed on the Smartcard. First of all, most of the applications use the cipher for the generation and verification of MACs and as a one-way function in the generation of session keys. In the cases where encipherment is actually used, the CFB or OFB mode can be used, where the inverse cipher is not used.

3.3 The key schedule

In a Smartcard implementation, computing the expanded key in a single shot and storing it for use during the actual encryption, would require too much RAM. Therefore, the key expansion has been defined in such a way that it can be implemented in a cyclic buffer with a size no larger than the size of the cipher key. The expansion operation has been kept very simple and efficient to make fast just-in-time key generation possible.

In the case that the cipher key length is equal to the block length, the round key is simply updated in between rounds. In the other cases, a small amount of additional sequence control is required. All operations in the key update can be efficiently implemented using EXORs and table lookups.

3.4 32-bit processors

On a 32-bit processor, an efficient implementation of the round transformation will use four large tables (1k each) that combine the effect of the nonlinear and the dispersion step. The tables will fit nicely in the cache of most modern 32-bit processors.

The performance is independent of the value of the multiplication factors in the dispersion step. The inverse operation of the cipher has the same performance as the forward operation, but uses a different set of tables.

4 Smartcard-specific attacks

Recently, several attacks have been demonstrated that exploit weaknesses of cipher implementations, rather than the inherent mathematical properties of the actual cipher [4]. These attacks exploit information such as timing and power consumption to obtain information on the cipher key or plaintext. In this section we will explain that the ciphers of the Square family lend themselves to implementations that provide resistance against this type of attack typical for Smartcards.

If programmed as explained in the previous section, the cipher execution consists of a series of instructions that is completely fixed: there are no conditional branches whose execution depends on the cipher key and input. This thwarts the following attacks:

Timing attack: This attack extracts key/plaintext information from the CPU time consumed by the cipher. For the Square ciphers the CPU time is independent of the cipher key and plaintext.

Simple power analysis: This attack extracts key/plaintext information by observing the power consumption during the cipher computation. Different CPU instructions have different power consumption and this attack allows to determine whether one or another conditional branch is taken in a computation. For the Square ciphers the series of instructions is fixed.

A more powerful attack is differential power analysis. This attack exploits the fact the power consumed by the CPU not only depends on the instruction that is executed, but also on the values of the operands. It combines the application of cryptanalytic techniques, statistics and power analysis. Basically, it allows the determination of the value of individual bits of intermediary computation results of the cipher by an attacker that does not even need to know the sequence of instructions. The basic flaw that is exploited is that the power consumed by an instruction depends on the values of the bits that are handled by the command. To thwart these attacks, two mechanisms are proposed:

scrambling: To complicate the exploitation of power consumption bias, e.g., by using programming tricks, such as insertion of a variable amount of NOPS, or better, unnecessary instructions in between rounds. Scrambling can be applied reasonably independent from the cipher structure.

curing: To make the power consumption of the relevant instructions used in a cipher

implementation much less dependent on the value of the treated bits. One plausible way to cure instructions is by the introduction of symmetry. For example: if a bit is stored in (loaded from) RAM, also store (load) its complement. If two bits are EXORed, execute all four different combinations: $a \oplus b, \bar{a} \oplus b, a \oplus \bar{b}, \bar{a} \oplus \bar{b}$. Typically, additional hardware has to be introduced for every sensitive instruction. Therefore it is an advantage to have few different instructions that handle key- or plaintext-dependent bits.

The instructions that handle key- or plaintext dependent bits in our implementations of the ciphers of the Square family are:

- EXOR with accumulator, direct addressing;
- store accumulator, direct addressing;
- load accumulator, direct addressing;
- load accumulator, indexed addressing (offset + index register)

These are only four instructions. Most other modern ciphers [5] use an instruction set that is substantially larger, due to the use of arithmetic operations. Moreover, the balancing of arithmetic operations is likely to be more complicated than the balancing of the EXOR. It can be seen that there is arithmetic addition in the indexed addressing. However, if the lookup tables are positioned at physical addresses that are a multiple of 256, the address computation can be reduced to a mere concatenation of index and offset. Obviously, this implies a modification of the ALU hardware.

5 Performance

We implemented the Square ciphers on two different types of microprocessors that are representative for Smartcards in use today. These implementations have been optimized towards minimal RAM usage and execution

time while guaranteeing a fixed execution time. Table 1 shows that besides the storage of the current round key and the intermediate ciphertext, only four bytes of RAM are used. The numbers compare very favorably with the figures for the other AES candidate algorithms.

The timings given include the key setup and algorithm setup time. Only the forward operation of the ciphers have been implemented, backwards operation is expected to be slower because the inverse of the diffusion step cannot be implemented as efficiently (cf. Section 3.2). The inverse diffusion step is between 1.5 and 2 times slower than the original diffusion step. Since the diffusion step is dominant in the execution of the ciphers on a Smartcard, about the same performance loss can be expected for the full cipher.

The implementations on the Motorola 68HC08 microprocessor have been done using the 68HC08 development tools by P&E Microcomputer Systems, Woburn, MA USA; the IASM08 68HC08 Integrated Assembler and the SIML8 68HC08 simulator. No optimization of code length has been attempted for this processor. Execution time, code size and required RAM for a number of implementations are given in Table 1 (1 cycle = 1 oscillator period = 0.25 μ sec).

Rijndael has also been implemented on the Intel 8051 microprocessor, using 8051 Development tools of Keil Elektronik: μ Vision IDE for Windows and dScope Debugger/Simulator for Windows. Execution time for several code sizes is given in Table 2 (1 cycle = 12 oscillator periods = 1 μ sec).

6 Availability

Several implementations of Square in C and Java are available from the URL <http://www.esat.kuleuven.ac.be/~rijmen/square>.

More information on Rijndael and a reference implementation are available from the URL <http://www.esat.kuleuven.ac.be/~rijmen/rijndael>.

Cipher (Key, block length)	Code size (bytes)	Required RAM (bytes)	Number of cycles	Execution time (msec)
BKSQ(96,96)	900	28	6500	1.6
Square(128,128)	919	36	6800	1.7
Rijndael(128,128)	919	36	8390	2.1
Rijndael(192,128)	1170	44	10780	2.7
Rijndael(256,128)	1135	52	12490	3.1

Table 1: Code size, required RAM and execution time for the square ciphers in Motorola 68HC08 Assembler.

(Key, block length)	Code size (bytes)	Number of cycles	Execution time (msec)
(128,128)	768	4065	4.1
	826	3744	3.7
	1016	3168	3.2
(192,128)	1125	4512	4.5
(256,128)	1041	5221	5.2

Table 2: Code size and execution time for Rijndael in Intel 8051 assembler.

References

- [1] J. Daemen, L.R. Knudsen, V. Rijmen, "The Square encryption algorithm," *Dr. Dobbs' Journal*, Vol. 22, No. 10, October 1997, pp. 54-56.
- [2] J. Daemen and V. Rijmen, "The Rijndael block cipher," presented at the First Advanced Encryption Standard Conference, Ventura (California), 1998, available from URL <http://www.nist.gov/aes>.
- [3] J. Daemen and V. Rijmen, "The Block Cipher BKSQ," *Proc. of CARDIS'98, LNCS*, Springer-Verlag, to appear.
- [4] P.C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 146-158.
- [5] NIST's AES Homepage:
<http://www.nist.gov/aes>.

PKCS #15 – A Cryptographic Token Information Format Standard

Magnus Nyström

RSA Laboratories, Bedford MA 01730, USA

E-mail: magnus@rsa.com

Abstract

We identify the need for a portable format for storage of user credentials (certificates, keys) on cryptographic tokens such as integrated circuit cards (IC cards). Given this need, a recent proposal in the area, RSA Laboratories' PKCS #15 is described and compared with previous and related work.

1 Background and Motivation

Cryptographic tokens, such as Integrated Circuit Cards (IC cards or "smart cards"), are capable of providing a secure storage and computation environment for a wide range of user credentials such as keys, certificates and passwords. Because of this, it is widely recognized (cf. [2], [15] and [25]) that they offer great potential for secure identification of users of information systems and electronic commerce applications. For a general introduction to IC cards and their use, cf. [3] or [17].

Unfortunately, the use of these tokens for authentication and authorization purposes is hampered by the lack of interoperability at several levels (cf. [1]). First, the industry lacks standards for storing a common format of digital credentials (keys, certificates, etc.) on them. This has made it difficult to create applications that can work with credentials from a variety of technology providers. Attempts to solve this problem in the application domain invariably increase costs for both development and maintenance. They also create a significant problem for end-users since credentials are tied to a particular application running against a particular application-programming interface to a particular hardware configuration.

Second, mechanisms to allow multiple applications to effectively share digital credentials have not yet reached maturity. While this problem is not unique to cryptographic tokens – it is already apparent in the use of certificates with World Wide Web browsers, for example – the limited room on many tokens together with the consumer expectation of universal acceptance will force credential sharing on credential providers. Without agreed-upon standards for credential sharing,

acceptance and use of them both by application developers and by consumers will be muted.

To optimize the benefit to both the industry and end-users, it is important that solutions to these issues be developed in a manner that supports a variety of operating environments, application programming interfaces, and a broad base of applications. Only through this approach can the needs of constituencies be supported and the development of credentials-activated applications encouraged, as a cost-effective solution to meeting requirements in a very diverse set of markets.

The purpose of the work we describe in this paper, PKCS #15 [19], has therefore been to:

- Enable interoperability among components running on various platforms (platform neutral);
- Enable applications to take advantage of products and components from multiple manufacturers (vendor neutral);
- Enable the use of advances in technology without rewriting application-level software (application neutral); and
- Maintain consistency with existing, related standards while expanding upon them only where necessary and practical.

By fulfilling these objectives, PKCS #15 is a first step to ensure that token-holders will be able to use their cryptographic tokens to electronically identify themselves to any application regardless of the application's token interface. The ultimate goal is a situation in which a token-holder can use any card from any manufacturer to identify himself or herself to any application running on any platform.

2 Related Work

2.1 DC/SC

"Digital Certificates on Smart Cards," DC/SC [1], was a collaborative effort mainly between CertCo, Litronics

and GemPlus, initiated to facilitate the interoperability of applications using digital certificates stored on IC cards. DC/SC concentrated on the problem of finding all digital certificates stored on a particular user's IC card. The proposed solution was to add an extra elementary file at the root level on the card system (ISO/IEC 7816-4 compliant IC cards were assumed), in which applications would read and write information about known certificates on the card.

The objective of DC/SC was very similar to PKCS #15's objective: To enhance portability of user credentials stored on IC cards. The main difference was perhaps that DC/SC only concentrated on certificates and did not intend to create a new card application for general credential storage. DC/SC eventually folded its work into the SEIS specification.

2.2 SEIS

SEIS, Secured Electronic Information in Society, is a Sweden-based non-profit association. One of SEIS' most important projects has been to specify an Electronic Identity IC card with an Electronic ID Application. This includes means for secure, electronic authentication of the cardholder; generation of legally acceptable Digital Signatures; and support for session key exchange, e.g. protection of message confidentiality. The intention is that these functions will be implemented using "off the shelf" IC cards. The usage is intended for security services both within, as well as between, organizations.

So far, two specifications have been developed which relate closely to PKCS #15: SEIS S1 [21] and SEIS S4 [22]. SEIS S1 is a detailed definition of an electronic identity card application. It specifies where and how information about certificates, keys and PINs shall be stored on a compliant IC card. SEIS S4 is a profile that puts some further restrictions on the format and the size of used keys.

Being a pure electronic identification application based on public-key technology, SEIS S1 and S4 do not contain support for any other key types than private RSA keys and supports only X.509 [10] certificates. Furthermore, there is no support for general security-related data objects.

Although it is more generic, PKCS #15 is a continuation of the SEIS work. The concept of a standardized, generic IC card format is a cornerstone of the SEIS architecture.

2.3 The WAP consortium

WAP, the Wireless Application Protocol Forum [12], is a consortium of companies involved in creating a de-facto standard for wireless information and telephony services on digital mobile phones and other wireless terminals. A part of this work is to enable secure identification of subscribers to these services. For this reason, subscribers will be issued IC cards (SIM cards, "Subscriber Identification Module"), which are to be inserted in phones, and the *WAP Identity Module Specification* (WIM) [26] has therefore basically the same objectives as PKCS #15. The current draft version, v0.7 is in fact designed as a PKCS #15 profile.

2.4 Other related work

This section contains a survey of other specification-, standardization- and product-efforts, related to PKCS #15.

2.4.1 The PC/SC specification

The *Interoperability Specification for ICCs and Personal Computer Systems* [20], or PC/SC for short, is a workgroup formed by leading IC card and personal computer vendors such as GemPlus, Schlumberger and Microsoft. The intention was to develop a specification that could facilitate the interoperability necessary to allow Integrated Circuit Card (ICC) technology to be effectively utilized in the PC environment, and in a manner which would "support both existing and future IC card-based applications" [20].

Part 8 of the specification, "Recommendations for ICC Security and Privacy Devices" [16], does mention (without binding requirements) a few formats for storage of information, but other than this, the PC/SC specification does not deal with the contents of the IC card itself. Hence, credential portability is not covered by the PC/SC specification. The intention with PC/SC is that each platform the user accesses with his IC card will have a *service provider* interface installed for that particular card. The drawback of this is that it forces cardholders to choose particular cards and vendors. One additional problem with this architecture is that it relies on the IC card vendor for providing both the card interface and the functional interface. An alternative solution would have been to separate this into two layers: one handling the card interface and one handling the format interface. With separated layers and an open card format standard, a generic PKCS#15 layer could be

installed and card layers added for each card the user/customer has. A user could choose any card type, have it personalized by any application vendor which personalizes cards in accordance with PKCS#15 and use it on any PC/SC system which has a PKCS#15 format service provider installed and, in addition, a service provider for that particular card type.

2.4.2 OpenCard Framework

The *OpenCard Framework* (OCF) [23] provides a common programming interface for both the smart card reader and the application on the card. By basing the architecture on Java technology, the intention is to receive enhanced portability and interoperability. The version 1.1 specification also enables, through an adaptation layer, interaction with existing PC/SC 1.0 supported reader devices.

Although the OCF simplifies the task for application builders, and has made explicit the distinction between a card layer and a format layer, it does not deal with the problem of credential portability. The *ApplicationManagement* layer is a rudimentary service layer in principle only supporting application listing and selection. OCF applications still need explicit knowledge of the location and structure of files contained in card applications (*card layout* in OCF terminology). Therefore, a cardholder will still be restricted to use his proprietary-formatted card only on those platforms, for which a card-application aware OCF application has been installed, which probably will be within a particular domain.

2.4.3 The JavaCard specification

The *JavaCard* specification [24], developed by Sun Microsystems, defines a subset of the Java language for use on IC cards and other embedded systems. The intention is to simplify card-application development by offering a familiar high-level programming language interface to developers of card-side applications. By implementing a version of the Java Virtual Machine on top of existing IC card operating systems, applications become portable and easier to develop.

The specification does not deal with card layouts or information formats, instead it defines a framework for creating applications. Even though it is possible to implement PKCS #15 on a JavaCard, it would probably be more natural to define a generic 'Electronic ID' JavaCard application (*cardlet* in established terminology). The JavaCard specification of such an

application would instead of specifying internal file formats define the command interface used to store and retrieve security-related information as well as to execute cryptographic commands.

2.4.4 MultOS

MULTOS [13] is another attempt to simplify card-side application development and portability of card-side applications. MULTOS is a card operating system that implements an Application Abstract Machine (AAM). The intention is that by using services offered by the AAM, application programmers do not need specific knowledge about underlying hardware.

Similar to the JavaCard case, it probably makes more sense to define a standard "MULTOS Electronic ID application", specifying the command interface rather than actual file formats in the MULTOS case.

3 An Overview of PKCS #15

Having described the problem background and related work, we now proceed to describe the standard itself, and requirements that have influenced its design.

3.1 Design Goals

Token neutral

- In order not to favor any particular brand or type of IC card, PKCS #15 has been designed in such a way that the standard can be implemented on any IC card with basic ISO/IEC 7816 compatibility. For example, no assumptions about IC card commands except those defined in ISO/IEC 7816-4 [4] have been made. In fact, it should even be possible to implement it on memory cards and software tokens.
- In order to support memory cards and tokens that do not have built-in support for encryption, provision has been made to allow stored objects to be encrypted and/or integrity-protected.
- Finally, in order to support tokens which do not have the notion of "files", PKCS #15 has been designed to allow all information to be stored in one contiguous block.

Standards compliant

- In order to be aligned with ISO/IEC 7816-5 [5] and ISO/IEC 7816-6 [6], the information format (as

seen by an interface communicating with the card) has been defined in ASN.1 [9].

- The format allows storage of information concepts such as *security environments*, defined in ISO/IEC FCD 7816-8 [7].
- The “object-oriented” approach chosen for PKCS #11 [18], treating keys, certificates and other data as objects with attributes and values, has been adopted for PKCS #15 as well. It has a proven record and eases PKCS #11-based implementations.

Self-contained

- Given an IC card with a PKCS #15 application on the chip, in order to be able to use the card for secure identification, it must be possible for a host-side application to find out which algorithms, keys and certificates that are present on the card. This led to the inclusion of *TokenInfo* files and *Object Directory* files, see Section 3.2.
- Applications also need to know how objects are protected, and procedures for accessing them. They also need to know which objects that are possible to update and which are not. This requirement has been met by introducing appropriate security attributes and *authentication objects* that can be referenced from protected objects like keys. Authentication objects are stored in *Authentication Object Directory* files, see Section 3.2.

Flexible and modular structure

- It should not be necessary to store all PKCS #15-relevant objects in the PKCS #15 application. Sometimes a certificate may already exist on the token when the PKCS #15 application is stored on it, for example. Therefore, an extra level of indirection has been introduced, giving the PKCS #15 application the ability to refer to objects like certificates stored in other dedicated files (or at other places like LDAP accessible directories).
- When dealing with IC cards, the problem of *card tearing* (cf. [3], pp. 175–176) has to be considered. This means that every update needs to be as atomic as possible, and if interrupted due to premature card removal, should not leave the card in an inconsistent state. This led to the decision to make the file structure record-oriented and modular.

- Since it is anticipated that PKCS #15 will be used in a number of applications for different purposes and with different security requirements, no particular requirements in terms of access restrictions have been mandated; an appendix giving general recommendations has been provided, however. This appendix also builds on ideas expressed in ISO/IEC CD 7816-9 [8].

3.2 File Structure and Motivations

The content of the PKCS #15 dedicated file is a bit dependent on the type of IC card and its intended use, but the following file structure is likely to be the most common, especially when the card is intended to be used for identification/authentication purposes:

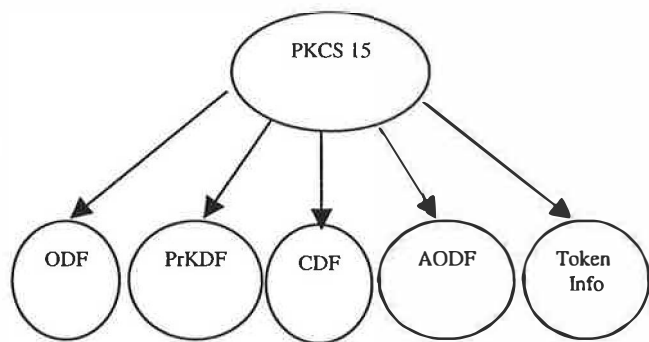


Figure 1 : Contents of DF(PKCS15)

The contents and purpose of major elementary files in the PKCS #15 directory are described below.

The Object Directory File, ODF

The mandatory elementary file ODF (Object Directory File) consists of pointers to other elementary files (PrKDFs, PuKDFs, SKDFs, CDFs, DODFs and AODFs), each one containing a directory over PKCS #15 objects of a particular class. The ODF therefore has a record-oriented structure, with each record merely being a pointer to another directory file.

Cryptographic Key Directory Files, PrKDFs, SKDFs and PuKDFs

These elementary files can be regarded as directories of keys known to the PKCS#15 application. PrKDFs contain information about private keys, PuKDFs contain information about public keys and SKDFs contain information about secret (symmetric) keys. They are all

optional, but at least one file of a particular kind must be present on an IC card which contains keys (or references to keys) of that particular kind, known to the PKCS#15 application. The files contain general key attributes such as labels, key usage restrictions, identifiers, type of algorithm, key size (if applicable), etc. Furthermore, they contain pointers to the keys themselves.

Certificate Directory Files CDFs

These elementary files can be regarded as directories of certificates known to the PKCS#15 application. They are optional, but at least one CDF must be present on an IC card which contains certificates (or references to certificates) known to the PKCS#15 application. They contain general certificate attributes such as labels, identifiers, certificate type, etc. They also contain pointers to the certificates themselves. When a certificate contains a public key corresponding to a private key that is also known to the PKCS #15 application, the certificate and the private key will share a common identifier. This simplifies look-up of the private key given the certificate and vice versa.

Trusted Certificate Directory Files

These elementary files have the same syntax as ordinary CDFs, but only contain trusted certificates. In the context of PKCS #15, "trusted certificates" are CA certificates not possible to be replaced by the cardholder.

Authentication Object Directory Files AODFs

These elementary files can be regarded as directories of authentication objects (e.g. PINs) known to the PKCS#15 application. They are optional, but at least one AODF must be present on an IC card, which contains authentication objects restricting access to PKCS#15 objects. They contain generic authentication object attributes such as (in the case of PINs) allowed characters, PIN length, PIN padding character, etc. Furthermore, they contain pointers to the authentication objects themselves (e.g. in the case of PINs, pointers to the directory in which the PIN file resides). Authentication objects are used to control access to other objects such as keys. Each object in this file has a unique reference number, which is used for cross-reference purposes from e.g. the PrKDFs to link keys with authentication objects.

Data Object Directory Files, DODFs

These files can be regarded as directories of data objects (other than keys or certificates) known to the PKCS#15 application. They are optional, but at least one DODF must be present on an IC card which contains such data objects (or references to such data objects) known to the PKCS#15 application. They contain general data object attributes such as identification of the application to which the data object belongs, whether it is a private or public object, etc. In addition to this, they contain pointers to the data objects themselves.

The TokenInfo file

The mandatory TokenInfo elementary file contains generic information about the token as such and its capabilities, as seen by the PKCS #15 application, e.g. supported algorithms, token serial number, etc. In order to save some storage space, provisions for cross-referencing algorithm information from the PrKDFs, PuKDFs and SKDFs to this file has been made.

4 An Example Application

PKCS #15 v1.0 contains a profile of the information format for electronic identification purposes. This profile specifies the use of two private keys, two user certificates and two PINs. Each private key is to be protected with a separate PIN and also linked to a corresponding certificate. The intention is that the cardholder use one key (and associated PIN) for general authentication purposes and key exchanges, and the other key (and associated PIN) strictly for non-repudiation (or digital signature) purposes, like signing documents.

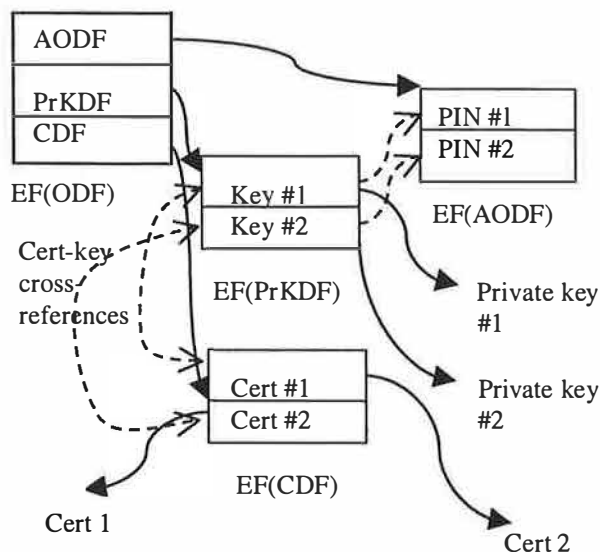


Figure 2: Logical file structure of PKCS #15's Electronic ID profile

If some 3rd-party application-specific data is added to a token containing the Electronic ID profile of PKCS #15¹, the token will be useable not only for general identification purposes but for that 3rd-party application's purposes as well. An example of this is the profile suggested in WIM [26], in which the cardholder will be able to use the card not only as a cellular phone card enabling phone calls and related services, but also enabling Internet access from any PKCS #15 aware application. If PKCS #15 is well received, it is likely that similar cases will occur in a number of other environments, like on-line banking, as secure credit cards, etc.

5 Summary and Future Work

PKCS #15 was announced in September 1998. Shortly thereafter, the SEIS specifications were adopted as national standards in Sweden. At the same time, the WAP Forum submitted its first Identity Module draft containing an IC card file structure. Coordinating with and using experiences from these efforts as well as other ones has been and continues to be an important part of this project. The first official version of PKCS #15 is expected to be available in April 1999.

The current draft version of the standard does not deal at all with more advanced IC cards like JavaCards or MULTOS cards. As mentioned in this paper, since these cards are able to store and execute more complex

applications, the equivalent of PKCS #15 in this environment would probably be a "standard electronic ID application," defining a service interface rather than an information format.

The current lack of standardization of security-related commands for IC cards presents a more serious problem with regard to interoperability. In order to achieve interoperability with PKCS #15, an application needs not only to have a format layer implementing support for the PKCS #15 format, but also a card adaptation layer, implementing support for proprietary security-related commands set for a range of IC cards. If or when a set of such commands becomes formally standardized and adapted by IC card vendors, this card adaptation layer would become superfluous. ISO/IEC 7816-8 [7] is intended to solve the problem of standardization in this area, and hopefully card vendors will adjust to this standard in a timely manner.

ISO/IEC JTC1 SC17 has recently discussed [11] a possible new work item defining an Electronic ID Application for identification cards. While the prospects for such a work item remain unclear, PKCS #15 could clearly be one candidate for this format. PKCS #15 has also been suggested as the card format for the national identity IC card in Finland [14].

6 Acknowledgement

The author wishes to acknowledge valuable suggestions and comments from Zoltan Kelemen, Security Dynamics, and Burt Kaliski, RSA Laboratories.

7 References

- [1] DC/SC, "Interoperability Specification for Digital Certificates – Storage of Digital Certificates on ICCs," draft version 0.2, Digital Certificates on Smart Cards Working Group, 1998.
- [2] S. Elliot and C. Loebbecke, "Smart card based Electronic Commerce: Characteristics and Roles," *Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS'98) (IEEE)*, 1998.
- [3] S. Guthery S and T. Jurgensen, *Smart Card Developer's Kit*, Macmillan Technical Publishing, 1998.

¹ Preferably in the form of PKCS#15 Data Objects.

- [4] ISO/IEC 7816-4, "Information Technology – Identification cards – Integrated Circuit(s) cards with contacts – Part 4: Interindustry commands for interchange," International Organization for Standardization, 1995.
- [5] ISO/IEC 7816-5, "Information Technology – Identification cards – Integrated Circuit(s) cards with contacts – Part 5: Numbering systems and registration procedure for application identifiers," International Organization for Standardization, 1994.
- [6] ISO/IEC 7816-6, "Information Technology – Identification cards – Integrated Circuit(s) cards with contacts – Part 6: Interindustry data elements," International Organization for Standardization, 1996.
- [7] ISO/IEC 7816-8 Final Committee Draft, "Information Technology – Identification cards – Integrated Circuit(s) cards with contacts – Part 8: Security related interindustry commands," International Organization for Standardization, 1998.
- [8] ISO/IEC 7816-9 Committee Draft, "Information Technology – Identification cards – Integrated Circuit(s) cards with contacts – Part 9: Additional interindustry commands and security attributes," International Organization for Standardization, 1998.
- [9] ISO/IEC 8824-1, "Information Technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation," International Organization for Standardization, 1997.
- [10] ISO/IEC 9594-8, "Information Technology – Open Systems Interconnection – The Directory: Authentication framework," International Organization for Standardization, 1997.
- [11] ISO/IEC JTC1/SC17, "Report of the 11th Plenary Meeting of ISO/IEC JTC1/SC17, Berlin, Germany, 1998-10-21/23," Document N1429, International Organization for Standardization, October 1998.
- [12] P. King, "The Wireless Application Protocol (WAP)," *Proceedings of RSA Data Security Conference '99*, San Jose, USA, 1999.
- [13] H. Kingdon, "MULTOS – The card for every lifestyle," *Proceedings of CardTech/SecurTech '98 West*, San Jose, USA, 1998.
- [14] M. Kontio, Personal communication, 1999.
- [15] T. Monk and H. Dreifus, *Smart Cards: A Guide to Building and Managing Smart Card Applications*, John Wiley & Sons, 1997.
- [16] PC/SC, "Interoperability Specification for ICCs and Personal Computer Systems – Part 8: Recommendations for ICC Security and Privacy Devices," The PC/SC Workgroup, December 1997 (Available from <http://www.smartcardsys.com>).
- [17] W. Rankl, W. Effing, *Smart Card Handbook*, John Wiley & Sons, June 1997.
- [18] RSA Laboratories, "PKCS #11: Cryptographic Token Interface Standard," version 2.01, December 1997 (Available from <ftp://ftp.rsa.com/pub/pkcs/pkcs-11>).
- [19] RSA Laboratories, "PKCS #15: Cryptographic Token Information Format Standard," version 1.0, April 1999 (Available from <ftp://ftp.rsa.com/pub/pkcs/pkcs-15>).
- [20] P. Sarlin, "PC/SC Technical Overview," *Proceedings of CardTech/SecurTech West*, San Jose, USA, 1996.
- [21] SEIS, "SEIS Cards – Electronic ID Application v2.0," The Association for Secured Electronic Information in Society, 1998 (Available from <http://www.seis.se>).
- [22] SEIS, "SEIS Cards – EID Implementation Profiles v2.0," The Association for Secured Electronic Information in Society, 1998 (Available from <http://www.SEIS.se>).
- [23] F. Seliger, "OCF: Java API für e-business Anwendungen mit Smart Cards", *Proceedings of OOP'99*, Munich, Germany, 1998.

- [24] Sun Microsystems, Inc., "The JavaCard 2.1 Platform Specifications," Sun Microsystems, 1999 (Available from <http://java.sun.com/products/javacard>).
- [25] E. Turban, D. McElroy, "Using Smart Cards in Electronic Commerce," *Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS'98) (IEEE)*, 1998.
- [26] WAP, "Wireless Application Protocol – Identity Module Specification – Part: Security", draft v0.7, Wireless Application Protocol Forum, 1999.

Remotely Keyed Encryption Using Non-Encrypting Smart Cards

Stefan Lucks Rüdiger Weis
Theoretische Informatik Praktische Informatik IV
University of Mannheim
68131 Mannheim, Germany
luck@th.informatik.uni-mannheim.de
rweis@pi4.informatik.uni-mannheim.de

Abstract

Remotely keyed encryption supports fast encryption on a slow smart card. For the scheme described here, even a smart card without a builtin encryption function, would do the job, e.g., a signature card.

1 Introduction

Many security relevant applications store secret keys on a tamper-resistant device, a *smart card*. Protecting the valuable keys is the card's main purpose. Typically, smart cards are slow. Using them for key-dependent operations such as en- and decrypting inherently must be slow as well, right? Wrong, paradoxically there is still a way of doing fast encryption using a slow card.

Often, smart cards are designed to support authentication or digital signatures instead of encryption. In this paper, we concentrate on the RaMaRK protocol, theoretically based on (pseudo)random mappings. Paradoxically enough: The RaMaRK protocol does not require the smart card itself to support encryption – support for hash functions, as built into many signature cards, is sufficient. In a world with lots of restrictions on the import, export or usage of encryption tools and much less restrictions regarding authentication or signature tools, this can be an important property.

2 Remotely Keyed Encryption

A *remotely keyed encryption scheme* (RKES) distributes the computational burden for a block cipher with large blocks between two parties, a *host* and a *card*. Figure 1 gives a general description of an RKES.

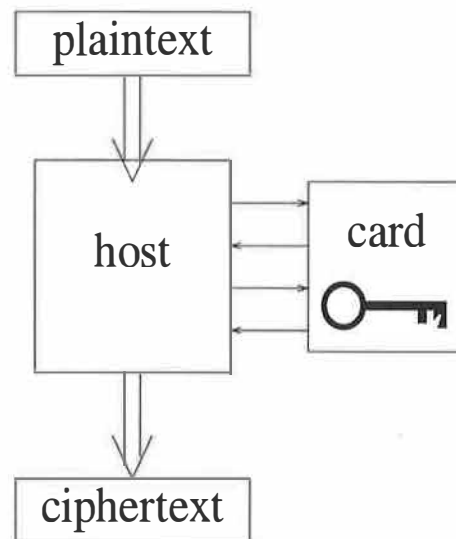


Figure 1: A generic RKES

We think of the host being a computer under the risk of being taken over by an adversary, while the

card can be a smart card, protecting the secret key. We do not consider attacks to break the tamper-resistance of the smart cards itself. The host knows plaintext and ciphertext, but only the card is trusted with the key.

An RKES consists of two protocols: the *encryption protocol* and the *decryption protocol*. Given a B -bit input, either to encrypt or to decrypt, such a protocol runs like this: The host sends a *challenge value* to the card, depending on the input, and the card replies a *response value*, depending on both the challenge value and the key.

The notion of *remotely keyed encryption* is due to Blaze [2]. Lucks [8] pointed out some weaknesses of Blaze's scheme and gave formal requirements for the security of RKESs:

- (i) *Forgery security*: If the adversary has controlled the host for $q-1$ interactions, she cannot produce q plaintext/ciphertext pairs.
- (ii) *Inversion security*: An adversary with (legitimate) access to encryption must not be able to decrypt and vice versa.
- (iii) *Pseudorandomness*: The encryption function should behave pseudorandomly for someone neither having access to the card, nor knowing the secret key.

While Requirements (i) and (ii) restrict the abilities of an adversary with access to the smart card, Requirement (iii) is only valid for *outsider adversaries*, having no access to the card. If an adversary could compute forgeries or run inversion attacks, she could easily distinguish the encryption function from a random one.

It is theoretically desirable, that a cryptographic primitive always appears to behave randomly for everyone without access to the key. So why not require pseudorandomness with respect to insider adversaries? In any RKES, the amount of communication between host and card should be smaller than the input length, otherwise the card could just do the complete encryption on its own. Since (at least) a part of the input is not handled by the smart card, and, for the same reasons, (at least) a part of the

output is generated by the host, an insider adversary can easily decide that the output generated by herself is not random.

In 1998, Blaze, Feigenbaum, and Naor [3] found another way to define the pseudorandomness of RKESs. Their formal definition is quite complicated. Is is based on the adversary A gaining direct access to the card *for a certain amount of time*, making a fixed number of interactions with the card. When A has lost direct access to the card, the encryption function should appear to behave randomly, even for A . Recently, Lucks [9] described an "accelerated" RKES, which satisfies the security requirements of Blaze, Feigenbaum and Naor, but is significantly more efficient. Note that both schemes actually require the card to execute encryption function, while this paper deals with remotely keyed encryption using non-encrypting smart cards.

Theoretically, one could define an encryption function based on random mappings and hence adapt the schemes of [3, 9] for the use of non-encrypting smart-cards. Such a construction could be based on using Luby-Rackoff ciphers [6], or on one of the many refinements of them, such as the one in [7]. In practice, the resulting RKE-scheme would be quite inefficient, though.

3 RaMaRK Encryption scheme

In this section, we describe the *Random Mapping based Remotely Keyed (RaMaRK) Encryption scheme*, which uses several independent instances of a *fixed size random mapping* $f : \{0,1\}^b \rightarrow \{0,1\}^b$. In practice, one uses pseudorandom functions¹ instead of truly random ones. The scheme is provably secure if its building blocks are, i.e., it satisfies requirements (i)–(iii) above, see [8]. Note that b must be large enough—performing close to $2^{b/2}$ encryptions has to be infeasible. We recommend to choose $b \geq 160$. By " \oplus " we denote the bit-wise XOR, though mathematically any group operation would do the job as well.

¹If f is "pseudorandom", it is infeasible to distinguish between f and a truly random function - except if one knows the secret key.

We use three building blocks:

1. Key-dependent (pseudo-)random mappings $f_i : \{0,1\}^b \rightarrow \{0,1\}^b$.
2. A hash function $H : \{0,1\}^* \rightarrow \{0,1\}^b$.
 H has to be *collision resistant*, i.e. it has to be infeasible to find any $t, u \in \{0,1\}^*$ with $u \neq t$ but $H(u) = H(t)$.
3. A pseudorandom bit generator (i.e. a “stream cipher”) $S : \{0,1\}^b \rightarrow \{0,1\}^*$. We restrict ourselves to $S : \{0,1\}^b \rightarrow \{0,1\}^{B-2b}$.

If the seed $s \in \{0,1\}^b$ is randomly chosen, the bits produced by $S(s)$ have to be indistinguishable from randomly generated bits.

In addition to pseudorandomness, the following property is needed: If s is secret and attackers choose $t_1, t_2, \dots \in \{0,1\}^b$ with $t_i \neq t_j$ for $i \neq j$ and receive outputs $S(s \oplus t_1), S(s \oplus t_2), \dots$, it has to be infeasible for the attackers to distinguish these outputs from independently generated random bit strings of the same size. Hence, such a construction behaves like a random mapping $\{0,1\}^b \rightarrow \{0,1\}^{B-2b}$, though it actually is a pseudorandom one, depending on the secret s .

Based on these building blocks, we realize a remotely keyed encryption scheme to encrypt blocks of any size $B \geq 3b$, see figure 2.

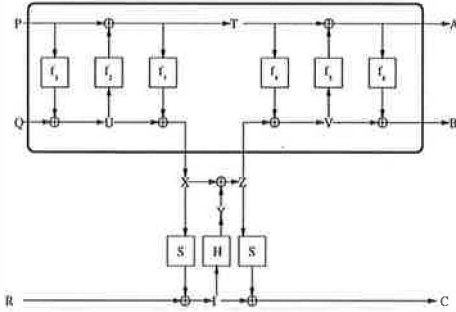


Figure 2: The RaMaRK encryption protocol

3.1 RaMaRK Encryption Protocol

We represent the plaintext by (P, Q, R) and the ciphertext by (A, B, C) , where $(P, Q, R), (A, B, C) \in \{0,1\}^b \times \{0,1\}^b \times \{0,1\}^{B-2b}$. For the protocol description we also consider intermediate values $T, U, V, X, Y, Z \in \{0,1\}^b$, and $I \in \{0,1\}^{B-2b}$. The encryption protocol works as follows:

1. Given the plaintext (P, Q, R) , the host sends P and Q to the card.
2. The card computes

$$U = f_1(P) \oplus Q \text{ and } T = f_2(U) \oplus P,$$

and sends

$$X = f_3(T) \oplus U$$

to the host.

3. The host computes

$$I = S(X) \oplus R \text{ and } Y = H(I),$$

sends

$$Z = X \oplus Y$$

to the card, and computes

$$C = S(Z) \oplus I.$$

4. The card computes

$$V = f_4(T) \oplus Z,$$

and sends the two values

$$A = f_5(V) \oplus T \text{ and } B = f_6(A) \oplus V$$

to the host.

3.2 RaMaRK Decryption Protocol

In order to decrypt the ciphertext (A, B, C) , we need the following protocol:

1. Given the plaintext (A, B, C) , the host sends A and B to the card.

2. The card computes

$$V = f_6(A) \oplus B \text{ and } T = f_5(V) \oplus A,$$

and sends

$$Z = f_4(T) \oplus V$$

to the host.

3. The host computes

$$I = S(Z) \oplus C \text{ and } Y = H(I),$$

sends

$$X = Z \oplus Y$$

to the card, and computes

$$R = S(X) \oplus I.$$

4. The card computes

$$U = f_3(T) \oplus X$$

and sends the two values

$$P = f_2(U) \oplus T \text{ and } Q = f_1(P) \oplus U$$

to the host.

One can easily verify that by first encrypting any plaintext using any key, then by decrypting the result using the same key, one gets the same plaintext again.

3.3 Remark

Neither the amount of communication between host and card, nor the amount of work on the size of the card depend on the block size B of the full cipher. Thus, if B is not too small compared to the parameter b , which defines the size of the blocks sent to the card, the RaMaRK scheme is efficient. The card itself operates on $2b$ bit data blocks, and both $3b$ bit of information enter and leave the card. In practice, $b \geq 160$ gives a high level of security, while B can be huge.

4 Extended Security Requirements

Regarding the RaMaRK scheme, the authors of [3] pointed out that an adversary A who had access to card and lost the access again, can later chose special plaintexts where A can predict a *part* of the ciphertext. This makes it easy for A to distinguish between RaMaRK encryption and encrypting randomly.² Thus, according to the definition of [3], the RaMaRK scheme is not pseudorandom.

We believe that it is possible to extend the RaMaRK scheme to make it pseudorandom even in the sense of [3], i.e., with respect to insider adversaries. So far, this is an open problem. Note that all schemes in [3] are pseudorandom as defined there, but depend on *pseudorandom permutations* (i.e., block ciphers) – and thus are designed for smart cards with builtin encryption.

5 Implementation of Building Blocks on the Host

On the side of the host, we need standard cryptographic primitive operations, which can easily be implemented or found in a cryptographic function.

5.1 Hash Functions.

To combine the big block of data with the small blocks in the card we need a collision-free hash function. The calculation is performed on the host, so we can simply chose a well-tested hash fuction like SHA-1[5] or RIPE-MD160[4]. Both produce a 160-bit output, which seems to provide sufficient security.

²The intermediate value X only depends on the (P, Q) -part of the plaintext, and the encryption of the R -part only depends on X . If A chooses a plaintext (P, Q, R) , having participated before in the encryption of (P, Q, R') , with $R \neq R'$, the adversary A can predict the C -part of the ciphertext corresponding to (P, Q, R) on her own.

5.2 Pseudo Random Bitgenerators.

In [8] the use of a stream cipher was suggested. We can also use a well-tested block cipher in the OFB or CFB mode (E.g. CAST-5 performs very fine even on small packets [10]).

6 Keydependent Pseudorandom Mappings on the Card

In this section we want to discuss how to realise Pseudo Random Mappings (PRM) with an Non-Encrypting smartcard. For the purposes of this paper, we suggest to use hash-based *Message Authentication Codes* (MACs) as tools. We specifically recommend the HMAC-construction from Bellare, Canetti, and Krawczyk [1], which is provably secure.

Note that a cryptographic hash function is defined to take a bit-string of an arbitrary length as input, to produce a fixed-size bit-string as output. (In addition to this, it also has to satisfy some cryptographic security criteria.)

6.1 Using a Hash-Based MAC to realize PRMs

Trusting in a well-studied dedicated hash function, such as SHA-1 or RIPEMD-160, to realize a key-dependent Message Authentication Code provides a couple of advantages for our scheme:

- Cryptographic hash functions have been well studied.
- Cryptographic hash functions are usually faster than encryption algorithms.
- MACs based on SHA-1 or RIPE-MD160 mostly provide 160-bit output. So even birthday attacks which need 2^{80} operations are infeasible.
- Some hash-based MACs are provably secure if the underlying hash is secure.
- The proof of security for some MAC constructions can rely on quite weak assumptions on the

hash function's security, compared to the standard assumptions for hash functions. Thus, even if the hash function we use is broken and insecure for signatures or other applications, it may still be infeasible to break the HMAC instantiated with this hash function.

- In many countries, it is more easy to export or import an authentication tool, such as a signature smart card, than to export or import an encrypting device, such as a smart card with a builtin encryption function.

6.2 HMAC: A Construction for Hash-Based MACs

HMAC [1] has the advantage that we can use any cryptographic hash function \mathcal{H} as blackbox. The only restriction on \mathcal{H} is the following: \mathcal{H} is assumed to be an *iterative* hash function. That means, it internally uses a compression function, iteratively taking a fixed-size value as input (say, 512 bit), to produce a smaller-sized output (e.g., 160 bit). Most cryptographic hash functions known today are iterative. If needed, one can easily define a secure iterative hash functions based on a secure non-iterative one.

The HMAC function is defined like this:

$$\text{HMAC}_K(x) := \mathcal{H}(\bar{K} \oplus \text{opad} || \mathcal{H}(\bar{K} \oplus \text{ipad} || x))$$

with $\text{ipad} := \text{Ox36}$ repeated 64 times and $\text{opad} := \text{Ox5C}$ repeated 64 times³, \bar{K} is generated by appending zeros to the end of K to create a 64 byte string⁴. (Note that the specific values of ipad and opad are relevant for actually implementing HMACs without creating incompatible versions, but with respect to the security of HMACs, one mainly has to keep in mind $\text{ipad} \neq \text{opad}$.)

In [1] a proof is given, that the HMAC construction is secure against collision attacks and forgery attacks.

³The number of repetitions may actually change, depending on the input size of the underlying compression function. Most present-day hash functions, including the well-studied SHA-1 and RIPEMD-160, use a compression function with an input size of 512-bit (i.e., 64 byte).

⁴This size of 64 byte also changes with the input size of the compression function

As usual in present-day cryptography, the proof of security is based on some unproven but reasonable assumptions. The weaker such assumptions are, the stronger is the proof. It is thus remarkable, that the proof in [1] only makes very weak assumptions on the security of the underlying hash function (and no assumptions otherwise).

Consider selecting a hash function \mathcal{H} for the HMAC construction, i.e., *instantiating* the HMAC construction with \mathcal{H} . Of course, this has to be done with great care. But it is the state-of-the-art in today's cryptography, that no one can rule out completely that this hash function \mathcal{H} is later found to be insecure, e.g., collisions for \mathcal{H} are found. A collision consists of two bit-strings $x \neq y$ with the same values, i.e., $\mathcal{H}(x) = \mathcal{H}(y)$. Such collisions do *exist* of course, but if it is feasible to actually *find* such collisions, this would be deathly for using \mathcal{H} in the context of signatures. On the other hand, due to the weak assumptions on \mathcal{H} the HMAC-construction requires, even a collision-prone hash function \mathcal{H} could still satisfy the security requirements for HMACs, and HMACs instantiated with \mathcal{H} could be secure, nevertheless.

References

- [1] M. Bellare, R. Canetti and H. Krawczyk, "Keying hash functions for message authentication" (1996), in: *Crypto 96*, Springer LNCS.
- [2] Blaze, M., "High-Bandwidth Encryption with Low-Bandwidth Smartcards", in: *Fast Software Encryption* (ed. D. Gollmann) (1996), Springer LNCS 1039, 33–40.
- [3] Blaze, M., Feigenbaum, J., and Naor, M., "A Formal Treatment of Remotely Keyed Encryption", in: *Eurocrypt '98*, Springer LNCS 1403, 251–265.
- [4] Dobbertin, H., Bosselaers, A., Preneel, B., "RIPEMD-160, a strengthened version of RIPEMD", *Proc. of Fast Software Encryption* (ed. D. Gollmann), LNCS 1039, Springer, 1996, pp. 71–82.
- [5] NIST, "Secure Hash Standard", Washington D.C., April 1995.
- [6] Luby, M., Rackoff, C., "How to construct pseudorandom permutations from pseudorandom functions", *SIAM J. Comp.*, Vol 17, No. 2, 1988, pp. 239–255.
- [7] Lucks, S., "Faster Luby-Rackoff ciphers", in: *Fast Software Encryption* (ed. D. Gollmann) (1996), Springer LNCS 1039.
- [8] Lucks, S., "On the Security of Remotely Keyed Encryption", in: *Fast Software Encryption* (ed. E. Biham) (1997), Springer LNCS 1267.
- [9] Lucks, S., "Accelerated Remotely Keyed Encryption", to appear in: *Fast Software Encryption* (1999), (ed. L. Knudsen) Springer LNCS, 1999.
- [10] Weis, R., Lucks, S., "The Performance of Modern Block Ciphers in JAVA", to appear in: *CARDIS'98*, Springer LNCS.

Smartcard Integration with Kerberos V5

Naomaru Itoi and Peter Honeyman
Center for Information Technology Integration
University of Michigan
Ann Arbor

itoi@eecs.umich.edu, honey@citi.umich.edu

Abstract

We describe our design and implementation of smartcard integration with Kerberos V5. Authentication is among the most important applications for smartcards and is one of the critical requirements for computer security. By augmenting Kerberos V5 with tamper-resistant hardware, we enhance the security of Kerberos V5 and offer a potential “killer application” leading to wider adoption of smartcard technology.

1 Introduction

Smartcards are a rapidly emerging technology that have received much attention both from industry and academia. Smartcards can make a significant impact on current computer systems because of their inherent security and mobility.

According to market researcher Dataquest, the smartcard market will grow from 544 million units in 1995 to 3.4 billion units by 2001. However, the vast majority of smartcards are used in Europe; 90 percent of worldwide smartcard shipments went to Europe in 1995. Only 2 percent were shipped to the Americas [4].

Smartcards are not popular in the United States because there is no “killer application” for smartcards here. Smartcards were introduced to Europe by government telecommunications monopolies in the form of phone cards,

but the telecommunications industry in the US is private and decentralized.

These cultural and economic differences are common to other smartcard applications prevalent worldwide, such as health care and banking. In addition, credit cards are more successful in the US than in Europe, in part due to online verification, which is universally available in the US. This keeps fraud rates low – reportedly 0.07% [11] – which allows card issuers to indemnify customers for any loss over 50 USD. Consequently, issuers, customers, and merchants are equipped and satisfied with magstripe cards and readers, which feature low cost and broad familiarity [3].

The information technology business sector might provide the killer application for the smartcard industry in the United States because the demand for secure computer environments is huge and growing. There is an increasing fear of hackers attacking sensitive information on the Internet. Smartcards can provide a secure authentication system when combined with sound authentication protocols, and can significantly improve computer system security wherever authentication plays a critical role in the computer security scheme, *i.e.*, everywhere.

At the University of Michigan, smartcards are already deployed and used for storing a small amount of cash. Thus, we have a good setting for extending the deployment of smartcards in the computer environment:

- Students and faculty are familiar with using smartcards.

- An infrastructure is well established, *e.g.*, many vending machines and shops have smartcard readers.
- There is a serious security problem that can be solved by integrating smartcards into the computer environment.
- University technologists, especially at the Center for Information Technology Integration (CITI), have skill sets and resources to develop smartcard applications.

The goal of our project is to develop, build, and deploy a smartcard-integrated computer environment. We want to provide a smartcard in everyone's pocket that handles computer authentication, computer profiles, electronic cash, banking, identification, course registration, paying rents, submitting resume, copy machines, *etc.* [6].

The centrally administered computing environment at the University of Michigan is protected by Kerberos, the most widely used network authentication protocol [21, 13]. Kerberos is also key to the security infrastructure at MIT (where Kerberos was invented), Cornell, Carnegie-Mellon, and Stanford, as well as in mainstream commercial product offerings from IBM, Microsoft, and Oracle.

Kerberos suffers from some inherent security pitfalls, principally its reliance on passwords selected by users. In recent years, CITI staff have used password guessing attacks [6, 7] on the University of Michigan Kerberos servers with (disappointing) success, quickly obtaining thousands of passwords on each occasion. To improve the security of Kerberos and the infrastructure it protects, we intend to replace passwords with randomly generated Kerberos keys stored on a smartcard.

The remainder of this paper is organized as follows. In Section 2, we describe why and how smartcards can enhance the security of Kerberos. In Section 3, we explain the protocol we use to integrate Kerberos with smartcards. Section 4 contains implementation details for those who want to port our program

to other operating systems or to use other types of smartcards. (Readers who are not interested in implementation details may want to skip the section.) Performance is evaluated in Section 5. Section 6 discusses related work and smartcards we have examined. Future directions are described in Section 7, followed by concluding remarks in Section 8.

2 How can smartcards help Kerberos?

Bellovin and Merritt enumerate problems of Kerberos that “are not solvable without employing special-purpose hardware, no matter what the design of the protocol.” [2] The problems include:

- Need for secure encryption device
- Need for secure key storage
- Dictionary attack on passwords

We explain these problems, and describe countermeasures that take advantage of strong security feature of smartcards.

2.1 Need for secure encryption device

In the Kerberos protocol, a user key, K_u , is shared between a user and a *Key Distribution Center* (KDC), a trusted third party. K_u is derived from a password: a workstation reads the password from a user, converts it to K_u , and uses it to decrypt a *ticket granting ticket* (TGT), an initial credential in Kerberos. The protocol is shown in Figure 1.

- 1) When a user attempts to login to a workstation, the workstation sends a request to the KDC.
- 2) KDC generates a TGT, encrypts it with K_u , and sends it back to the workstation.
- 3) The workstation asks the user for a password, hashes it into key K_u , and uses the key

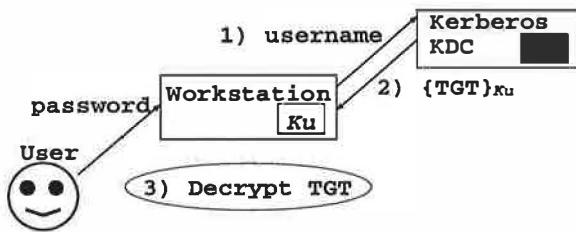


Figure 1: Kerberos authentication protocol without a smartcard

to decrypt the TGT. If the TGT decrypts properly, the user is authenticated and is allowed to login.

In this protocol, K_u is exposed to two parties, a user and a workstation. A key memorized by a user can be vulnerable because she can tell it to another person, or an adversary might “shoulder surf” it when she types it. If she is using a window manager, her keystrokes may even be snooped remotely without her knowledge.

A key in a workstation can be vulnerable if the workstation is not securely protected or cannot be trusted for other reasons. For example, if an adversary can scan the entire physical memory of the workstation, he can obtain the key. Along the same lines, if someone has administrative access rights to the workstation, it is straightforward to install a rogue login program in the workstation that stores a user’s password in the adversary’s directory. (This is called a Trojan horse attack.)

To solve these problems, it is desirable to decrypt the TGT outside a workstation. Therefore, an external encryption device is required.

2.2 Need for secure key storage

Kerberos stores some keys in computers, *e.g.*, session keys in a workstation and user keys in KDC. However, typical computers cannot store information securely. Information in a computer system is stored either in memory or on a hard disk, but neither is sufficiently secure. A secret stored on a hard disk is hard to protect

because:

- A powerful adversary can access (read and write) it.
- It is usually backed up in mass storage devices, which may lack sufficient physical or cryptographic protection.

A secret in memory is also hard to protect because :

- Memory can be physically scanned by a powerful adversary.
- It may be paged out to hard disks, which can be scanned.

Therefore, secure storage outside a workstation and KDC is an important goal.

2.3 Dictionary Attack

When a user chooses a poor password, the derived user key K_u , is subject to dictionary attack. Dictionary attack is performed as follows:

1. Create a list of common words, names, etc.
2. Derive keys from the words in the list.
3. Obtain a <plaintext, ciphertext> pair.
4. Decrypt the ciphertext with the derived keys.
5. If the plaintext is recovered correctly, the key used for decryption is revealed.

For example, if the password is a short English word, an adversary can try all English words in the dictionary and quickly discover the password.

Kerberos is vulnerable to dictionary attack because:

1. It is a password-based authentication protocol.
2. It easily gives up a <plaintext, ciphertext> pair to the adversary.

Test runs of dictionary attack in the University of Michigan Kerberos realm have yielded passwords for more than 5% of the user accounts, *i.e.*, over 4,000 accounts [6].¹

To solve problem (2), pre-authentication is introduced in Kerberos V5. The Kerberos authentication protocol with pre-authentication is depicted in Figure 2.

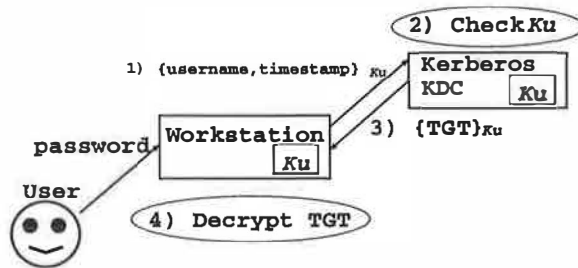


Figure 2: Kerberos authentication protocol with pre-authentication

In this scenario, the KDC ensures that the client knows K_u before issuing a TGT. 1) When the client requests the TGT, it sends a username and a timestamp encrypted with K_u . 2) If the KDC can successfully decrypt with K_u and recover the username and a valid timestamp, it is sure that the client knows K_u . If not, the KDC assumes someone is impersonating the client to obtain a <plaintext, ciphertext> pair and rejects the request. 3) After pre-authentication, the KDC sends the TGT encrypted by K_u to the workstation and the protocol continues as depicted in Figure 1.

Pre-authentication prevents an adversary from getting a <plaintext, ciphertext> pair just by requesting it, and thus raises the bar of security to the adversary. However, the adversary can still eavesdrop a network to obtain a <plaintext, ciphertext> pair. Also note that it is very easy for the adversary to recognize a

¹The most common password was "love". Go figure.

plaintext because it includes well known entries such as a user name and a realm name.

As long as Kerberos uses passwords for secure information, dictionary attack cannot be solved completely. Therefore, it is desirable to replace passwords with randomly generated bits stored in tamper-resistant hardware [17].

A smartcard is an ideal device to solve the problems outlined here. The countermeasures are described in the next section.

3 Design

In this section, we describe a method intended to enhance the security of Kerberos. It takes advantage of a smartcard to solve the problems stated in Section 2.

From the discussion in Section 2, our design goals are:

- Use randomly generated bits for K_u .

We can prevent dictionary attack by using a random key instead of a user chosen password. However, we then require a way for users to store their keys, as it is impossible (and insecure!) to expect anyone to remember a random string of any substantial size.

- Store a user key in a smartcard.

A smartcard can serve as an external key store because it is designed to be tamper-proof with restricted communication mechanisms.

- Decrypt TGT in a smartcard.

A smartcard can perform decryption as an external encryption device because it has DES en(de)cryption mechanisms.²

- Do not modify KDC.

²Or claims to. Many smartcards claim to offer DES but they in fact do not. We discuss this further in Section 6.2

If KDC must be modified to implement the smartcard augmentations, then our efforts will offer enhanced security in our local Kerberos realm, but nowhere else. We also want our improvements to enhance the security of Kerberos realms beyond our administrative control.

3.1 Protocol

Figure 3 shows our Kerberos authentication protocol with a smartcard. Steps 1) and 2) are identical to the original protocol (Figure 1). 3) When the workstation receives the TGT, it does not decrypt it by itself. Instead, it sends the TGT to a smartcard. 4) The smartcard then decrypts the TGT, and returns the TGT in plaintext to the workstation. 5) If the workstation confirms that the decrypted TGT is correct, the protocol is finished and the user is authenticated.

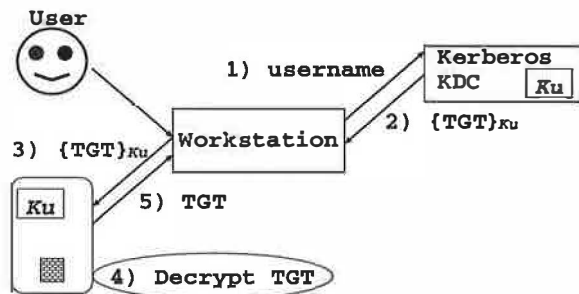


Figure 3: Kerberos authentication protocol with a smartcard

The protocol satisfies the goals we stated above; TGT is decrypted in the smartcard, K_u never leaves the smartcard, K_u can be random bits, and KDC is not modified.³ Furthermore, use of a smartcard obviates the requirement for preauthentication; even if plaintext, ciphertext, pairs are obtained through network snooping, the keys are selected at random, so are immune to dictionary attack

³In fact, KDC in Kerberos V5-1.0.5 must be modified by one line to run the protocol due to a bug in Kerberos. However, this modification will not be necessary in later version of Kerberos. We discuss it in Section 4.1.

4 Implementation

We implemented the smartcard integrated Kerberos protocol described in Section 3. We now detail the modifications we made to the Kerberos library, the DES library, and the Kerberos client.

TGT decryption is implemented with two smartcards:

- STARCOS version 2.1 from Giesecke & Devrient
- Cyberflex Access from Schlumberger

Both cards provide native *cipher block chaining* (CBC) for long messages. (A Kerberos V5 TGT is over 200 bytes long.) Cyberflex Access is a Java programmable card. STARCOS is not programmable.

The development platform is OpenBSD-2.4 on Pentium 400MHz PC. The code base is Kerberos version 1.0.5 released by MIT.

4.1 Adding an encryption system in Kerberos library

Kerberos V5 uses a look-up table to provide for easy replacement and development of encryption systems [12]. The look-up table associates an encryption type to cryptographic functions, such as encryption, decryption, and checksum functions, and data structures, such as a key structure. It is simple to add a new encryption system entry by adding an entry to the look-up table.

There are several encryption system types defined in the RFC[12] and implemented in Kerberos V5-1.0.5 including:

- No encryption
- DES in CBC mode with a CRC-32 checksum (`des-cbc-crc`)

- DES in CBC with MD5 (`des-cbc-md5`)

We created a new encryption system, DES in CBC with MD5 with a smartcard (`des-cbc-md5-sc`). We added a new entry `des-cbc-md5-sc` in the look-up table. The entry is defined in `des_md5.c` (Figure 4).

```
krb5_cryptosystem_entry
mit_des_md5_sc_cryptosystem_entry {
    EncryptionType ENCTYPE_DES_CBC_MD5_SC;
    DecryptionFunc mit_des_md5_sc_decrypt_func();
    // Other members are identical to des-cbc-md5
};
```

Figure 4: Smartcard cryptosystem entry

`mit_des_md5_sc_decrypt_func()` is a new function that uses a smartcard for decryption. The other members of the entry are not modified.

Although the default hash method in Kerberos V5-1.0.5 is CRC, implementation of `des-cbc-crc` in Kerberos V5-1.0.5 has a bug. In the Kerberos 5 specification, the *initialization vector* (IV) of DES-CBC mode is defined to be 0 [12]. However, `des-cbc-crc` uses the key as the IV. This error can not be fixed easily because Kerberos 5 is already deployed widely and several commercial offerings use the key as the IV as well. The G&D smartcard cannot use the key as an IV without passing it as an argument to the card, which defeats our goal of eliminating the key on the workstation.

To our relief, `des-cbc-md5` uses 0 as the IV, complying with the RFC. Rumor has it that the next version of Kerberos will use `des-cbc-md5` by default.

4.2 Modifying DES library

`mit_des_md5_sc_decrypt_func()` calls the DES-CBC encryption function in `f_cbc.c`. We created a new DES-CBC function `mit_des_cbc_sc_encrypt()` that calls a DES function in a smartcard instead of a Kerberos DES library. STARCOS version 2.1 can handle up to 112 bytes in one command. The TGT,

whose length is approximately 200 bytes, is divided into two pieces, decrypted in a smartcard piece by piece, and combined into one TGT in the workstation.

The specific commands, or APDUs in ISO 7816-4, sent to the smartcard are as follows. (Readers not familiar with smartcard APDUs are advised to consult the ISO 7816-4 specification [8] or Guthery and Jurgensen's book [5].)

- Send “decrypt” APDU with 112 (0x70) bytes of encrypted data.

0x80 0xf8 0x81 0x81 0x70 data ...

- Send “get response” APDU to upload 112 bytes of plaintext data.

0x00 0xc0 0x00 0x00 0x70

We repeat these steps with the second half of the TGT, using an IV taken from the first half.

4.3 Modifying kinit

In the authentication function `get_in_tkt()`, an encryption system can be chosen as an argument. We modified `kinit.c` so that it does not request a password from the user, and specified encryption type `des-cbc-md5-sc` instead of `des-cbc-md5`.

5 Performance Evaluation

Here we evaluate the performance of our Kerberos modifications.

5.1 Performance Evaluation

We ran the authentication protocol described in Section 3.1 by executing the modified `kinit`

program five times and logged salient performance data. The authentication time fluctuates within a relatively small range (about 0.1 sec.), averaging 1.14 sec. with STARCOS and 2.96 sec. with Cyberflex. STARCOS communicates at 115 Kbps, and Cyberflex at 56 Kbps. We analyze performance in detail in the following sections.

5.2 Time line

Figure 5 shows a time line of the kinit program.



Figure 5: Time line

- 1) start kinit program, 2) start RPC to Kerberos KDC, 3) end RPC, do some host pre-processing, 4) initialize smartcard (reset, set baud rate, select key file), 5) call decryption routine in smartcard, 6) store a ticket in a file, and do some post-processing.

5.3 Breakdown

Table 1 shows how much time is spent in each part of the protocol. Time is in seconds.

part	STARCOS	Cyberflex
pre-processing (1,2,3)	0.158	0.307
smartcard time (4,5)	0.944	2.617
– initialize card (4)	0.243	0.358
– data transfer	0.053	0.100
– decryption	0.323	1.047
– misc	0.325	1.113
post-processing (6)	0.039	0.037
total	1.139	2.961

Table 1: Authentication time breakdown

Total time to authenticate with the STARCOS card is 1.139 sec. (2.961 sec. for CyberFlex Access). Smartcard-related tasks – initialization, communication, and decryption – domi-

nate, taking 83% (88%) of the total time. With an 8-bit data path and a 3.5 MHz clock, a smartcard is much slower than a workstation.

The rest of time, including RPC communication with KDC, is 0.197 sec. (0.344 sec.). Of the 0.944 sec. (2.617 sec.) of smartcard time, decryption takes 0.323 sec. (1.047 sec.), initialization (card reset, set baud rate, select key file) takes 0.243 sec. (0.358 sec.), data transfer takes 0.053 (0.100) sec., and miscellaneous part takes 0.325 (1.113) sec. The miscellaneous part includes APDU handling overhead.

As a non-programmable card, STARCOS shows better performance than Cyberflex Access, but not dramatically so. For STARCOS, we send APDUs that invoke the built-in DES-CBC method, while for Cyberflex, we coded and loaded a Java applet that receives a request APDU and then calls a built-in DES-CBC routine. We think these two cases illustrate the tradeoff between performance and flexibility available to smartcard developers.

6 Discussion

6.1 Related Work

Here we relate this effort to secure computer systems, secure bootstrapping, smartcard authentication, and smartcard integration with Kerberos.

Secure computer system

We refer to two efforts that share our goal of building a secure computer environment.

In the Dyad system [22], Tygar and Yee build an operating system for a tamper-proof coprocessor that has the ability to process and store secrets. The coprocessor provides secure bootstrapping, secure logging, and copy protection.

The Dyad approach is top-down: unlike most

security protocols, Tygar and Yee do not assume security of components of computers such as operating systems because an adversary can reload the kernel. To address this problem, they build special purpose hardware and an operating system for it. This approach differs markedly from ours.

Another related, top-down approach is described by Lampson *et. al.* [14], who develop a theory of authentication for distributed systems based on an access control model. They build tools necessary for secure systems, such as encrypted channels, boot strapping, naming, and program loading. Accompanying the design of these tools are formal proofs of their security. Finally, they build an operating system to take advantage of the tools.

Both Dyad and TAOS take top-down approaches: they start with a well-developed theoretical framework, then design secure hardware to support the theory, then build operating systems based on them.

Although these approaches are substantive and technically sound, they are not practical for most existing computer environment because they build new operating systems from scratch. We take a more pragmatic and experimental approach and build from the bottom-up for rapid implementation and deployment. We employ currently available, secure, inexpensive hardware in the form of commercial smartcards, integrate them with prevalent standards, and fit them with minimal effort into our existing computer environment.

A disadvantage of our approach is that we still rely on the security of hardware and operating systems, of which we cannot be sure. (Often, we have great doubts!) For example, if an operating system is completely replaced, it is quite possible for an adversary to use stolen credentials to access resources.

Our solution to this problem is to store all critical secrets in a smartcard. A smartcard is tamper-resistant hardware, so no matter what happens to the hardware and the operating system, we can be confident that the secrets in

the smartcard remain safe. In the previous example, even if the operating system is compromised, critical information in a smartcard, such as authentication keys, can not be accessed by the adversary. Therefore, our approach significantly “raises the bar” of security in a computer system with relatively small cost.

Secure Bootstrap

Arbaugh *et. al.* introduce AEGIS, a secure bootstrap process [1]. They add a small PROM to commodity hardware. The PROM is assumed to be secure, *i.e.*, it is not replaced by the adversary. The PROM contains execution code to start bootstrapping and to check digital signatures. During the bootstrap process, all execution code is verified by a digital signature. At the end of the bootstrap process, a commodity operating system, FreeBSD in their example, starts up. As the execution code in PROM and the bootstrap process are trusted, the operating system is trusted when it starts.

AEGIS is similar to our approach in the sense that both try to minimize components that must be trusted: the added PROM in AEGIS, and the smartcard in our case. Also, both use commodity hardware and software. AEGIS and our approach complement one another because AEGIS aims at starting an operating system securely, and we aim at establishing a secure computer environment built on top of operating systems.

Authentication with Smartcards

Several authentication protocols that use smartcards have been proposed. For example, Rubin proposes one-time password [18], Shoup and Rubin propose session key distribution in the third-party setting [20], Leach proposes the use of zero knowledge authentication [15], and Wang and Chang propose use of public key authentication in smartcards [23]. Each of these concentrates on one-to-one authentication, such as when a user logs in to a computer. This differs from our approach in that we in-

tegrate a smartcard into a standard authentication protocol already in heavy use. Among them, only Shoup and Rubin's protocol has actually been implemented with a smartcard [10].

Smartcard Integration with Kerberos

Looi *et. al.* describe smartcard integration with SESAME, which is compatible with Kerberos V5 [16]. Their approach is very similar to ours. They describe two ways of accomplishing smartcard integration:

1. Store a user key in a smartcard, load the key into a workstation, and use it for decrypting TGT instead of a derived key from a password.
2. Decrypt TGT in a smartcard.

Method 1 is not as secure as method 2 because the user key is loaded in a workstation. If the workstation is not trusted, the key is vulnerable. For example, a Trojan horse attack can easily obtain the key. Method 2, identical to our method, had not been implemented at the time of their writing.

6.2 DES in Smartcards

Many vendors claim that their smartcards support DES, but we had a very hard time getting a smartcard that meets our requirements, even though all we need is pure, unadulterated DES. Here we list some of the DES-capable smartcards that let us down when examined closely:

- Schlumberger CryptoFlex
Only custom-made cards have DES.
- Schlumberger MultiFlex
DES is available in the form of an internal authentication command, which returns only six the eight bytes of output data.

- IBM MFC

The smartcard encrypts a random number challenge presented by SCT_CMD_AUTHENTICATE command, but does not document a general-purpose DES interface.

- MAOSCO MULTOS

The card supplied with the developer's kit encrypts with a fixed key, **0x41ad8223a90be2a1**. According to the manual, "for security reasons," DES uses a "known cryptographic key." (!)

- General Information Systems OSCAR

The DES key is XOR'ed with a random number before it is used. According to their e-mail: "The keys are XOR'ed with a random number for security reasons." While this may help secure the serial link between the terminal and the reader, it makes the card useless for enterprise security deployment.

- Gemplus GPK

The key size is limited to 40 bit, a flaw not shared by Kerberos.

Eventually we found smartcards that satisfy our needs: Giesecke & Devrient STARCOS and Schlumberger Cyberflex Access.

7 Future Direction

Comparison among several smartcards

We plan to implement the Kerberos authentication protocol in more smartcards, *e.g.* IBM MFC, MULTOS, and so on.⁴ We expect to find some differences in their performance because:

- Some of the smartcards have DES CBC mode.

⁴If we receive smartcards with DES. See our discussion in Section 6.2.

- Some of the smartcards have key scheduling APIs.
- Communication speed differs among smartcards.

We also expect to find differences in user friendliness and stability among smartcards and developer's kits.

Kerberos tickets in a smartcard

As we argued in Section 2, it is desirable to store keys in a smartcard rather than in a workstation. Therefore, storing session keys in addition to the user key in a smartcard adds security to the protocol. If tickets are stored on a smartcard, it is secure to leave a workstation to have a cup of coffee as long as the user brings the smartcard with her. Although an adversary can access the console, he cannot access resources protected by Kerberos because he does not have session keys.

Smartcard integration with PAM and NT-PAM

We will address secure single sign-on. Combined with PAM [19] or Windows NT-PAM [9], smartcards can provide secure single sign-on [7] because they can store keys and passwords securely, and can be integrated into existing authentication protocols, as we have shown in this paper.

8 Conclusion

In this paper, we identified certain limitations of Kerberos and ways that a smartcard can counter them. We suggested a protocol that takes advantage of the secure features of a smartcard to enhance security of Kerberos. The protocol is implemented with a Giesecke & Devrient STARCOS smartcard and Kerberos

V5-1.0.5. Performance evaluation shows the protocol runs reasonably fast.

Acknowledgment

We thank Andrew Webb and Giesecke & Devrient America, Inc. for providing us with STARCOS smartcards and the smart tools to use them effectively. Timothy M. Jurgensen, Dave Sims, and K. Krishna at Schlumberger provided us with Cyberflex Access smartcards and answers to many questions. Jim Rees and Kevin Coffman at CITI provided valuable technical assistance.

This work was supported by Schlumberger's Program in Smartcard Technology at CITI.

References

- [1] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. A secure and reliable bootstrap architecture. In *1997 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.
- [2] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. In *Proceedings of the Winter 1991 Usenix Conference*, January 1991. ftp://research.att.com/dist/internet_security/kerblimit.usenix.ps.
- [3] Jorge Ferrari et al. *Smart Cards: A Case Study*. IBM Redbook, 1998. <http://www.redbooks.ibm.com/SG245239/sg245239.htm>, Section 1.11.
- [4] Smart Card Forum. factoids. <http://www.smartcrd.com/info/more/Factoids.htm>.
- [5] Scott B. Guthery and Timothy M. Jurgensen. *Smart Card Developer's Kit*. MacMillan Technical Publishing, Indianapolis, Indiana, December 1997.

- [6] Peter Honeyman. Ubiquitous smart-cards at the University of Michigan. [http://www.citi.umich.edu / projects / sinciti / smartcard / smartcard-vision.html](http://www.citi.umich.edu/projects/sinciti/smartcard/smartcard-vision.html), 1997.
- [7] Peter Honeyman, William A. Adamson, and Jim Rees. Joining security realms: A single login for Netware and Kerberos. In *Proceedings of Fifth USENIX UNIX Security Symposium*. USENIX, June 1995. Salt Lake City.
- [8] The International Organization for Standardization and The International Electrotechnical Commission. *ISO/IEC 7816-4 : Information technology - Identification cards - Integrated circuit(s) cards with contacts*, 9 1995.
- [9] Naomaru Itoi and Peter Honeyman. Pluggable authentication module for Windows NT. In *Proceedings of 2nd USENIX Windows NT Symposium*, Seattle, August 1998. USENIX.
- [10] Rob Jerdonek, Peter Honeyman, Kevin Coffman, Jim Rees, and Kip Wheeler. Implementation of a provably secure, smartcard-based key distribution protocol. In *CARDIS'98*, Louvain-la-Neuve, Belgium, Sept. 1998. Third Smart Card Research and Advanced Application Conference.
- [11] VISA Ken Ayer. Standardization in chip card security evaluations. Presentation in SCIA workshop, November 1998.
- [12] John T. Kohl and B. Clifford Neuman. The Kerberos network authentication service (V5), September 1993. Request For Comments 1510.
- [13] John T. Kohl, B. Clifford Neuman, and Theodore Y. T'so. The evolution of the Kerberos authentication system. *Distributed Open Systems*, pages 78-94, 1994. IEEE Computer Society Press.
- [14] Butler Lampson, Martin Abadi, Machael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. In *Operating Systems Review*, volume 27-5, pages 165-182. ACM, December 1993.
- [15] John Leach. Dynamic authentication for smartcards. *Computers & Security*, 14(5):385-389, 1995.
- [16] Mark Looi, Paul Ashley, Loo Tang Seet, Richard Au, Gary Gaskell, and Mark Vandenwauver. Enhancing SEMAME V4 with smart cards. In *CARDIS'98*, Louvain-la-Neuve, Belgium, Sept. 1998. Third Smart Card Research and Advanced Application Conference.
- [17] Joseph N. Pato. Using pre-authentication to avoid password guessing attacks, 1993. OSF DCE Request For Comments 26.0.
- [18] Aviel D. Rubin. Independent one-time passwords. *USENIX Journal of Computer Systems*, February 1996.
- [19] V. Samar and R. Schemers. Unified login with pluggable authentication modules (PAM), October 1995. OSF Request For Comments 86.0.
- [20] Victor Shoup and Avi Rubin. Session key distribution using smart cards,. In *Proceedings of Eurocrypt '96*, pages 321-331, Saragossa, Spain, May 1996.
- [21] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 USENIX Conference*. USENIX, February 1988.
- [22] J. D. Tygar and Bennet Yee. Dyad: A system for using physically secure coprocessors. Technical report, Carnegie Mellon University, May 1991. CMU-CS-91-140R.
- [23] Shiuh-Jeng Wang and Jin-Fu Chang. Smart card based secure password authentication scheme. *Computers & Security*, 15(3):231-237, 1996.

Mutual Authentication with Smart Cards

Bastiaan Bakker <Bastiaan.Bakker@Lifeline.nl>
Delft University of Technology, the Netherlands

Abstract

The World Wide Web has become the de facto interface for consumer oriented electronic commerce. So far the interaction between consumers and merchants is mostly limited to providing information about products and credit card based payments for mail orders. This is largely due to the lack of security currently available for commercial transactions. At the moment the only security mechanism present in most browsers is the Secure Socket Layer (SSL) which is limited to authentication and encryption of the HTTP session. It does not aim to secure transactions.

This report describes the design of a new three party authentication and key distribution protocol to serve as a foundation for WWW based transactions. Instead of having a radically new design it is derived from KryptoKnight protocol family developed at IBM. An important design consideration has been that it can be implemented with existing smart card technology. Specifically the Dutch Chipper and ChipKnip cards have been examined for their applicability. The result is an ABK(t) type protocol that runs with any card that supports either the ISO7816 *internal authenticate* command or the En726 *read stamped* or *protected read* instructions.

Secondly a prototype has been implemented in Java that can run in either the Java Development Kit or the Netscape or HotJava browser. Though Java was not designed for implementing hardware drivers it has proven perfectly suitable for communication with smart cards. Also it has effectively demonstrated its cross platform capabilities over multiple operating systems: except for a small native library to talk to the RS232 port the same code runs on Win32, Linux and the NCD network computer.

1. Introduction

The subject of this graduation assignment is to design and prototype a mutual authentication system for WAN-based Client/Server applications. Implementation of the system shall be based on common Internet technology. More specifically the applications are assumed to be WWW enabled, with a Java capable WWW browser on the client side and an HTTP server on the server side.

The system shall provide the basic elements for the initialization of a secure channel from client to server: authentication and (session) key negotiation. Properties

of the channel it self, such as encryption, message integrity, and non-repudiation, etc., are beyond the scope of this assignment.

Some countries, notably the USA and France pose restrictions on the export of strong encryption technology. In order to accommodate unrestricted development in these countries the authentication system shall be built only with cryptography that may be exported legally.

Since the primary users of the system are consumers ease of use is very important. Consumer oriented systems for securing transactions that rely on soft keys instead of hardware tokens, such as the IPay system have proven to be difficult to use and administer. Hardware tokens on the other hand store the keys and the algorithms that may use them internally, safe from harm by careless users or hard disk crashes. Therefore this assignment will investigate the possibilities to use hardware tokens as security providers. Particularly smart cards will be examined because they are designed for this purpose. Moreover since the introduction of two nation wide electronic purse systems, the Chipper and the ChipKnip, by all Dutch consumer banks the majority of Dutch consumers owns one or more smart cards.

For reasons of cost and availability it is attractive to be able to reuse these existing smart cards to authenticate clients and servers to each other. Typically the server is not the card issuer or owner of part of the card and therefore has no knowledge of the keys used by the smart card of the client. One option would be to install a security module containing all relevant keys at each server. This approach is used for the aforementioned ChipKnip and Chipper electronic purse implementations. This was motivated largely by the restriction that the purse transactions could be performed off line, without intervention of a central system. However in case of transactions on the Internet offline processing is not required. When considering the security and management issues generated by massive deployment of security modules, using a central security server is a preferable alternative. Consequently the authentication protocol features three parties: a client (the consumer), a server (the service provider) and a trusted third party (the smart card issuer, e.g. a bank).

The core of the system is a new cryptographic protocol that can be implemented with smart cards. It shall provide:

- Source authentication. The protocol should return an identifier of the peer party. This identifier may be persistent, such as an account or membership number or a transient, valid only for the duration of the session. In case of persistent IDs the trusted third party guarantees that the presented IDs belong to the communicating party. The protocol shall fail to complete if either party is not authenticated.
- Key negotiation. If authentication is successful, the protocol shall return a shared secret to both communicating parties. This secret can be used as a session key to provide message authentication, message integrity and optionally encryption.

1.1. Elements

Within an authentication system one can identify several elements. These include the users of the system of course, others are:

- Identifiers. These are the objects the users (both clients and servers) use to prove their identity. They contain an identifier for their owner and a mechanism to authenticate this identity. In this assignment the application of smart cards as identifiers will be investigated.
- A Trusted Third Party (TTP). The issuer of the identifiers. It is trusted by both client and server parties, hence its name.
- Semi Trusted Computing Base. The to be secured application runs in this environment. The semi trusted computing base is trusted not to compromise the security of a single session of the application, but not trusted enough to protect long term secrets used for identification.

1.1.1. Requirements on the Identifiers

In physical life people prove their identity with an identification paper, for example a passport. In 'computer life' we'll need a similar *identifier*.

Passports and alike contain three parts:

- An organization wide identifier: a social security number, a name, age, birthplace, birthday tuple, etc. This information is used within the 'organization' to uniquely refer to a person.
- Biometric identification information: photographs, age, height, eye color, etc. This data set should uniquely describe anyone bearing a passport and thereby link a passport to one unique individual.
- Physical authentication proofs: watermarks, uncopiable prints, special paper, etc. They should prove the passport is genuine and actually issued

by the claimed authority (and thereby prove that this authority attests to the identity of the bearer).

So passports rely on biometric verification and the unfeasibility to physically forge them. Biometric verification relies on the trusted verifier (e.g. a customs officer) to be present at the same place as the identifier (passport): the biometric properties are public, their security lies in their unforgeability.

Since we are looking for a distributed protocol the peer party cannot know whether properties are actually measured or just pretended to be measured.

Challenge Response Identifiers

Instead of relying on publicly known properties that may be forged, the identifying properties should be kept secret. You prove your identity by showing you know a unique secret that no other user knows, without disclosing it. A common method for this is a challenge/response protocol: party A sends a random number (the challenge) to party B, whose identity has to be verified. B returns a response calculated from the challenge and its unique secret key. The calculation has the property that it is infeasible to determine the key from the challenge response pairs. Nor is it possible to deduce new pairs from known ones. Verification of the response is only possible for parties knowing the secret used in the calculation. They can reproduce the challenge/response calculation and verify the response given by the authenticating party.

Physical, electrical and functional properties of smart cards are standardized in ISO standard 7816 ([ISO89], [ISO92] and [ISO93]). The majority of currently deployed cards implement 7816 parts 1 to 3 and most application level commands in part 4. The latter part includes some standardized commands to perform challenge/response calculations. The actual choice and implementation of a particular algorithm is left to the card manufacturer.

Minimally we require the card to contain an identifier and a key that are unique for the owner of the card and maintained by the Trusted Third Party (TTP). Examples of the identifiers are (bank) account numbers, social security numbers, membership numbers, etc. Secondly we require the card to offer some challenge response mechanism using the unique key that can prove the validity of the ID supplied by the card.

1.1.2. Requirements on the Trusted Third Party

The Trusted Third Party is the organization issuing the smart cards or at least controlling the part of the card used for the protocol in case of Multi Function Cards

(MFCs). It has knowledge of the persistent keys used in the protocol. As the name suggests, the client and server trust the TTP to provide them accurate information. Also they trust the TTP to issue identifiers that have sufficient security provisions to keep the persistent keys secret.

1.1.3. Requirements on the Semi Trusted Computing Base

A user may consider his or her computer and web browser to be a semi trusted computing base. People trust their computer and browser to communicate with the service provider, that is they trust it not to compromise the communication, for example to relay it to another party, to manipulate it, etc.

However Internet enabled computers are compromised on a daily basis all over the world, so one cannot rule out the possibility a hacker gains control over the web browser or even the whole computer. An important feature of the authentication system has to be that in such a case compromise will be incidental and not total: once the security breach in the semi trusted base has been fixed compromise should no longer be possible. Note that this is different from a breach in the security of the trusted computing base: once a smart card has leaked its secret nothing can be done to restore the security of that card.

1.2. Standardized Cryptographic Features for Smart Cards

Both the ISO 7816 and En726 standards only include commands that use secret key cryptography. Consequently their authentication mechanisms require that both parties (the card and the application) know the authentication key.

ISO 7816 specifies a single command by which a card can authenticate it self to the outside world: the *internal authenticate* command. This is a straightforward challenge/response mechanism with challenges and responses of typically eight bytes (this is not mandatory though). Commonly this command is implemented with DES encryption in Electronic CodeBook (ECB) mode. Conversely the *external authenticate* command allows the outside world to authenticate it self to the card.

ISO 7816 also specifies mechanisms for providing data integrity and confidentiality, labeled Secure Messaging. These do not seem to be implemented widely though, so they will not be discussed here.

Additionally the En726 standard offers methods for authentication and concealment of data on the card. Elementary files can be given access conditions, which specify whether and how the files may be read and/or updated. Among the conditions are "ALW" (for

always), "NEV" (for never), "PRO" (for protected) and "ENC" (for enciphered). The "PRO" access condition mandates that a cryptogram shall be sent along with the data read from or written to the card. This cryptogram is the result of a keyed MAC over the data and a challenge. The "ENC" access condition is an extension of the "PRO" condition. It mandates that all data shall be enciphered as well as protected by a cryptogram.

1.3. Key Management of Some Smart Card Systems

Electronic purses like Chipper and ChipKnip are designed for off line use: the customer should be able to pay the merchant without the need for a connection to a central server of a bank. This means only two parties are involved: just the card of the customer and the Payment Terminal System (PTS). Generally the part of the terminal that provides the security (contains all keys, etc) called the Secure Application Module (SAM), is a smart card as well.

Secret key based authentication requires both parties to share a common secret. The simplest method to arrange this is to have a single organization wide key stored in all smart cards. In closed systems deployed within a single organization, authentication with a single key is still commonly used. An example is the Closed Electronic Purse on the Studenten ChipKaart of 1995/96 [Hoekstra97].

For larger systems such as a nation wide intersector electronic purse a single global authentication key is not a feasible solution: breaking the security of a single smart card, that is obtaining the key, compromises the security of the entire purse application. Failure of a single entity in the system results in failure of the security of the entire system.

To remedy this electronic purses like Chipper and ChipKnip feature a unique authentication key per purse card. Obtaining the key of one purse card does not help you to forge other purse cards. After a couple of fraudulent purchases with a forged version of the cracked card the bank will notice the fraud (because more money was 'downloaded' from the card than was previously uploaded to it) and black list the card. The solution is only partial however: since every Payment Terminal System (PTS) should be able to authenticate every purse it should know all authentication keys. It is infeasible to store every key in a big database located in the PTS; we would be talking about something like 5 million entries replicated in at least 50.000 databases. Instead the authentication key of a purse is determined by key transformation: the bank has a single master key that serves as a key encrypting key or key generating key over the ID number of the card: $K_{\text{purse}} = E_{K_m}(\text{ID}_{\text{purse}})$ or $K_{\text{purse}} = \text{MAC}_{K_m}(\text{ID}_{\text{purse}})$. Now the SAM only needs

to know the master key from which it can deduce the purse key for every purse it has to authenticate. The purse itself contains only its own K_{purse} . Of course if this master key extracted from a SAM somehow, the entire system still is compromised completely.

2. A New Protocol for Authentication & Key Distribution

The KryptoKnight protocol family provides three party based authentication and key distribution built on exportable symmetric cryptography. It has most of the properties we desire. Nevertheless it has not specifically been designed for implementation with existing smart cards. For example it assumes party A and B know their respective keys K_A and K_B . For smart cards this is not true: neither smart card users nor applications are allowed to know the keys stored in the card. The operating system of the card only permits applications to initiate certain commands that use the key, but these are all executed within the secured environment of the card itself.

This chapter describes the design of a three party authentication and key distribution protocol suitable for smart cards, based on the KryptoKnight protocol designs. More specifically the ABK variant of the KryptoKnight protocol will be adapted to a smart card compatible ABK(t) type protocol, named KLOMP (KryptoKnight-based Lightweight Open Mutual authentication Protocol).

2.1. Assumptions and Constraints

1. Only the server shall contact the trusted third party.
2. Assumptions about the identifiers used by the client and the server shall be minimized.
3. The protocol shall be compatible with contemporary smart card technology.
4. The protocol shall be suitable for a Wide Area Network (WAN) environment.
5. All cryptographic algorithms shall be legally exportable (out of the United State and other countries that put restrictions on export of cryptography).
6. The protocol shall be stateless with respect to the trusted third party.
7. The trusted party shall not need to give any response before successful authentication of both client and server. Specifically it shall not disclose any information about the client to the server before both client and server are authenticated and the client has given its consent.

Ad 1) In a typical Internet based client/server system the number of clients is much larger than the number of servers. Furthermore the relation between a trusted party

and the application servers is more static than between the trusted party and the clients. Therefore both from a security and a topological viewpoint it is wise to limit access to the trusted party to the application servers.

Ad 2) Minimizing the assumptions about identifiers maximizes the reusability of the protocol.

Ad 3) The protocol is specifically aimed at letting smart cards provide the security for it. It shall at least be implementable with ISO7816 or En726 compliant cards.

Ad 4) The protocol is aimed at use in an Internet environment, which certainly is a Wide Area Network.

Ad 5) Authentication and key distribution by themselves are exportable functions. In many cases they suffice and confidentiality by encryption is not needed for the subsequent transactions. With careful design it is possible to build a system that is both secure and does not rely on export restricted cryptography

Ad 6) A stateless protocol considerably reduces the amount of administration to be kept by the trusted party. Keeping the server of the trusted party simple is important:

- compared to the other parties, the authentication server has to handle many requests: whereas the client has to perform only one authentication, the application server has to perform as many authentications as there are clients connecting and the trusted third party has to serve requests from all the applications servers it serves.
- a simple server is more robust than a complex one.
- a simple server is more likely to be secure than a complex one.

Ad 7) To ensure the privacy of users any information about them should be disclosed (to the application server) only after the user agrees to it, which means that both the user and the server should be authenticated first. Also for security it better not to have to send encrypted messages (tickets) to unauthenticated principals: it allows attackers to collect cipher texts which might help breaking the cryptographic protocols.

2.2. Generating MACs with smart cards

The KryptoKnight protocol needs to calculate authentication codes over messages longer than 32 bytes. Smart cards may not support MAC generation over messages of that length. They often offer only limited support for generation of authentication codes over messages. Some methods are:

- ISO7816 Internal Authenticate
- En726 Read Stamped / Protected Read of freely writeable field
- En726 Read Stamped / Protected Read of read only field
- CBC-MAC based of one of the above methods.

2.2.1. ISO7816 Internal Authenticate command

The ISO7816 Internal Authenticate command returns a keyed hash of a short message (the challenge) in order to authenticate the card to the outside world. This keyed hash can also be used to generate an authentication code over an arbitrary message. Since the challenge that the Internal Authenticate command accepts must have a limited length (typically 8 bytes), the message first has to be compressed using a secure hash algorithm. The compressed message is then fed to the Internal Authenticate command. In order to ensure freshness of the resulting Message Authentication Code, a random challenge number should be included in the message before compression. To put it in a formula: $MAC_K(M) = InternAuth_K(H(R // M))$, where M is the message, K the authentication key and R a random number to ensure freshness.

Although this method does yield a MAC generating algorithm, the compression of the message before the Internal Authenticate command does weaken it. For example in order to find two messages that collide (generate the same MAC) an attacker does not need to have access to the Internal Authenticate command. She only needs to find collisions in the secure hash algorithm. Any collisions found will then occur with any smart card regardless of the key.

2.2.2. En726 Read Stamped of Freely Writeable Field

The smart card supports the En726 *Read Stamped* or *Protected Read* methods and has a freely writeable file that can be read with (one of) these methods. This offers the best way to perform MAC calculation on the card: first write the message on the card and then read it back with the *Read Stamped* or *Protected Read* method. The only limitation is that the message has to fit into the file.

2.2.3. En726 Read Stamped of Read Only Field

In this case the smart card supports the *Read Stamped* or *Protected Read* methods but none of the files on the card is freely writeable. This gives the same opportunities as the *Internal Authenticate* command: an 8-byte challenge that yields an 8-byte response. The only difference is that the contents of the file have to be

sent to the verifier along with the response or else the verifier will not be able to reproduce the MAC calculation.

2.2.4. CBC-MAC

All of the above methods use an authentication code generating command only once. Depending on the speed of the card and the time constraints on the protocol this may be the maximum as well. But if one is permitted multiple invocations, either of the methods mentioned above can be used to build a CBC MAC, analogous to the ANSI X9.9 keyed MAC algorithm.

The Read Stamped over Writeable Field and the CBC-MAC method generate MACs over the entire message, the others only over a compressed representation of the message, which is less secure, so the first are preferred. But since they might not always be available or feasible the protocol will be designed to work with the other two methods as well.

2.3. Challenges, responses and authentication proofs

From the previous paragraph follows that minimally available is a challenge/response mechanism that accepts fixed length short challenges (of 8 bytes in most cases) and returns responses of the same length. In other words where KryptoKnight calculates MACs with K_A or K_B on complete messages, the new protocol first has to compress these messages. With a secure hash function the message can be mapped to a fixed length challenge that appears pseudo random. For example let $C_A = H(N_A, N_B, B)$ and $C_B = H(N_A, N_B, A)$.

However this introduces a possible weakness: the space of possible challenges is much smaller. When the space is small enough it might be feasible for an attacker that has already has collected correct challenge/response pairs to try to find a N_A (or N_B) that yields a known challenge by brute force. For example if the challenge is 64 bits big and the attacker has collected 2^{16} challenge/response pairs, every one in 2^{48} nonces yields a challenge for which the response is known. With the currently available computing resources this is feasible by 'brute force'. This attack is possible because the search for a suitable nonce can be performed off line. In order to limit this off line search the protocol may impose a time window outside of which the challenge is not valid. A window that is small enough renders a brute force attack infeasible: once a suitable challenge has been found the transaction has already expired. The challenges become:

$C_A = H(N_A, N_B, T, B)$ and $C_B = H(N_A, N_B, T, A)$, where T is a time stamp.

2.4. Order and Direction of the Data Flow

The order in which the three parties communicate is largely dictated by the constraints put on the protocol: the limitation that the protocol should be stateless with respect to the TTP (constraint 6) implies that it shall be contacted only once during a protocol run. The contactor shall be the server (see constraint 1). So the protocol flow includes $B \rightarrow K \rightarrow B$.

Since A has to be notified about success or failure of the protocol B subsequently sends a message to A. And since the ID of A has to be known before contacting the protocol flow has to be at least $A \rightarrow B \rightarrow K \rightarrow B \rightarrow A$. From constraint 7 follows that both the client and server shall generate their authentication proofs before contacting the TTP. This in turn implies that the challenges the client and server identifier have to respond to cannot include a nonce from the TTP. Instead it has to be substituted by a time stamp. In order to establish challenges that contain nonces of both A and B and the time stamp, the protocol flow has to start with a message from B to A that contains B's ID and its nonce N_B . If the protocol will be initiated by A it has to send a (possibly empty) 'trigger' message to B before that. This results in the following protocol flow: $A \rightarrow B \rightarrow A \rightarrow B \rightarrow K \rightarrow B \rightarrow A$.

All in all the desired protocol can be called an ABK(t) protocol, using the naming convention of the KryptoKnight documentation. (The t after the K for the KDC = TTP, indicates the substitution of a timestamp for a nonce from the TTP.) Unfortunately the KryptoKnight family does not include a protocol of this type, so a new one has to be developed.

2.5. Session Key Distribution

The method of key distribution is also influenced by the constraints on the protocol. The KryptoKnight protocols all let the TTP generate the session key and distribute it with tickets to both the client (A) and server (B). As discussed before this has the drawback that an attacker may collect any number of tickets for an attack on the encryption algorithm without any authentication.

In the new algorithm a different distribution mechanism is proposed: party A generates the session key, transports it encrypted (via party B) to the TTP, which decrypts the key and finally sends it to party B reencrypted with a key known to B. So in this case the session key is transported (from A to B) rather than distributed (from the TTP to A and B).

A and B generate cryptographic proofs before contacting the trusted third party. In the same effort the session key can be generated as well: party A generates session key K_{AB} with the same parameters as proof P_{AK} it sends to the TTP. Since these parameters are either publicly sent

to the TTP or known by the TTP (in case of the key transforming key K_A), the TTP can reproduce the session key without further knowledge.

2.6. Protocol Description

The considerations in the previous paragraphs lead to the protocol flows depicted in figure 1.

Step 1 might seem strange since it does not include any protocol data. Its purpose is to trigger the start of the authentication protocol and need not be explicitly performed: the protocol might be triggered implicitly when the client requests a secured action within the application protocol.

In step 2 the server replies with an identifier for itself (B), a nonce (N_B) and a time stamp (T). T substitutes for a nonce N_K of the TTP because the TTP may be contacted only once, after the challenge/responses have been performed at the client and server. The time stamp does not need to be very precise: it sets but a time window in which the authentication session has to be performed. It is checked by the TTP, so the clocks of the server and the TTP must have a time skew less than the allowed time window. A time window in the order of 5 to 10 minutes seems reasonable: it is small enough not to compromise the security of the protocol and wide enough to avoid the difficulties of tightly synchronized the two clocks.

Steps 3 and 4 are split up in three parts: In step 3a the client calculates a challenge C_A to send to the identifier. This challenge should be a secure hash of N_A , N_B , T and B, where N_A is a nonce generated by the client. Proposed is $C_A = \text{MAC}_T(N_A // N_B // B)$.

In step 3b the client collects the data from the identifier: the ID of the client (A), the type of the identifier (I_A), the response on the challenge (K_{A*}), and optionally any extra data used in the calculation of the response (D_A). Rather than having the TTP generate and distribute the session key, response K_{A*} will be used as a temporary key for the generation of session key K_{AB} . The choice for key generating function $f()$ depends on whether party B is allowed to learn challenge/response pairs of A's smart card. In that case $f(x)=x$ will suffice else an (optionally salted) hash function has to be used. The authentication proof P_{AK} is calculated from A, N_A , N_B and K_{AB} using formula $P_{AK} = \text{MAC}_{AB}(N_A // N_B // A)$. Using K_{AB} instead of K_{A*} makes the proof from A for the TTP identical to the proof from A for B: $P_{AK} = P_{BK}$.

In step 3c the client sends all parameters (A, N_A , I_A , D_A , P_{AK} , B, N_B , T) to party B.

The server (party B) similarly generates a challenge (C_B) and retrieves the corresponding response (K_{B*}) from its smart card. However the server does not need to generate a session key nor should its authentication proof be verifiable by the client. Therefore it can directly apply response K_{B*} in its calculation of P_{BK} . Since the server does not know K_{AB} yet it cannot verify P_{AB} . Instead the server sends all parameters ($A, N_A, I_A, D_A, P_{AK}, B, N_B, I_B, D_B, P_{BK}, T$) to the trusted third party.

In step 5 the TTP verifies all parameters received from B:

- is time stamp T valid?
- is A a valid and active client ID?
- is B a valid and active server ID?
- corresponds proof value P_{AK} to the value yielded with genuine key of the client K_A ?
- corresponds proof value P_{BK} to the value yielded with genuine key of the client K_B ?

In case the authentication request passed verification, the TTP calculates the session key and encrypts it for the server using key encrypting function $g()$: $T_{BA} =$

$g(K_{B*}, K_{AB}, \dots)$. if A may learn K_{B*} the function may be very simple again: $g(x, y, \dots) = x \oplus y$. The TTP sends back the result, the ticket T_{BA} , to the server B .

In step 6 the server decrypts the session key K_{AB} . Now B is able to verify proof P_{AK} , because P_{AK} is calculated from secret K_{AB} and the public values N_A, N_B and A . If the verification either client A or the trusted third party are imposters or the communication has been tampered with. If verification is successful B generates proof P_{BA} for client A and sends it to A . Finally A verifies proof P_{BA} .

2.7. Improvements on the protocol

The basic KLOMP protocol has been designed to be implementable with any challenge/response capable identifier. With some identifiers the protocol can be enhanced for better security. At the cost of some extra calculations also some potential weak spots can be eliminated.

2.7.1. Collection of challenge / response pairs

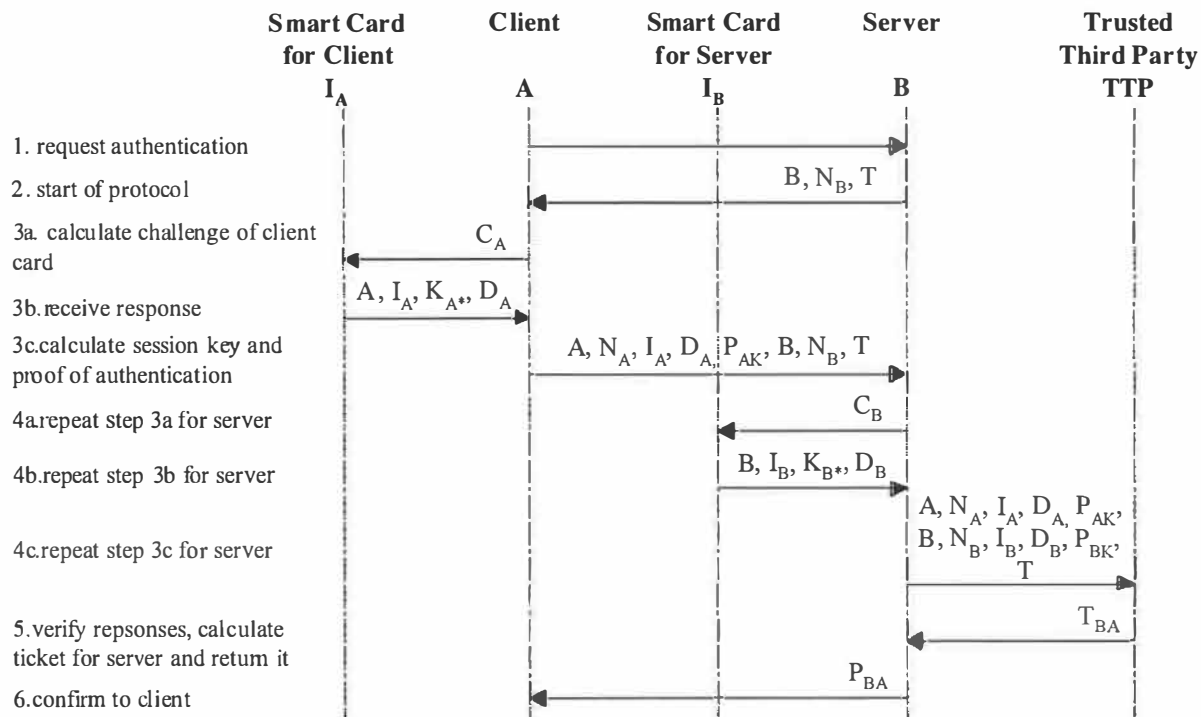


Figure 1: Data flow of the KLOMP protocol

$$\begin{aligned}
 C_A &= H(N_A, N_B, T, B) \\
 K_{A*} &= \text{MAC}_A(C_A, D_A) \\
 K_{AB} &= f(K_{A*}, \dots) \\
 P_{AK} &= P_{AB} = \text{MAC}_{AB}(N_A // N_B // A) \\
 T_{BA} &= g(K_{B*}, K_{AB}, \dots)
 \end{aligned}$$

$$\begin{aligned}
 C_B &= H(N_A, N_B, T, A) \\
 K_{B*} &= \text{MAC}_B(C_B, D_B) \\
 P_{BK} &= \text{MAC}_{B*}(N_A // N_B // B) \\
 P_{BA} &= \text{MAC}_{AB}(N_A // N_B)
 \end{aligned}$$

The current protocol allows servers to collect challenge/response pairs of the identifier of the client. Conversely clients can collect pairs of the identifier of the server if they eavesdrop on the connection between the server and the trusted third party. The reason for this weakness is obviously the simplicity of the key encrypting and key generating function applied in the protocol. To remedy this let:

$$K_{AB} = H(K_{A*} // N_A) \text{ and } T_{BA} = H(K_{B*} // N_B) \oplus K_{AB}$$

2.7.2. Clients searching for suitable C_A 's

If a rogue client knows some challenge/response pairs of the client identifier it may try to find a challenge he knows by trying different client nonces (N_A 's). Of course the search must be completed before the time stamp T expires. In any case this attack can be prevented if the client already has to commit to a specific N_A in step 1. A way to accomplish this is to let the client send a concealed commitment to the server in step 1: it has to send a verification parameter $V_A = H(N_A, S_A)$. Here S_A denotes the salt used to randomize the hash, analogous to the salt in encrypted UNIX passwords. The salt is a public value, meaning that the client sends it together with V_A to the server. The inclusion of the salt in the hash increases the input address space of the hash beyond the point where the server can deduce N_A through a dictionary attack. So the secure hash and the salt prevent the server to deduce the challenge of the client before committing to the value of its own nonce N_B . Yet in step 3c the server can verify that the client already has chosen its nonce before it knew the server nonce, thereby eliminating the search for a suitable nonce.

2.7.3. Time Window of Session Key Compromise

The session key K_{AB} may be used not only to protect the integrity and origin of transactions between A and B but also to encrypt their communication. If the information exchanged between A and B has to stay confidential even after the session has ended, K_{AB} must remain secret as well. Since K_{AB} is calculated solely from long term key K_A and data susceptible to eavesdropping, compromise of K_A will enable an attacker to recalculate all session keys, no matter how long ago the corresponding sessions took place (provided that he recorded those sessions of course). The same applies to the compromise of K_B , since it forms the basis for the transport key for K_{AB} . In case of key distribution protocols implemented completely in software, like KryptoKnight, the best we can hope for is that K_A nor K_B are ever revealed to other parties: there is no solution for this problem.

However with hardware security tokens like smart cards the situation may be much better. Compromise can be divided in two levels now:

1. An attacker obtains access to the smart card and particularly to the challenge/response generating command: this means that the attacker can obtain responses over arbitrary challenges.
2. An attacker discovers the key (K_A or K_B) used by the challenge/response algorithm on the card.

The second level equals the compromise of the key in an implementation in software only: the attacker can recover any session key. Luckily level 2 compromises are very unlikely to occur: smart cards are specifically designed to withstand any attempt (both logically and physically) to illegitimately retrieve the keys they contain.

Level 1 compromises are far more likely to happen, in fact any stolen or lost card may be abused for it. Luckily these compromises are also less severe. The key authentication and distribution protocol does not take advantage of the different properties of level 1 and level 2 compromises: in both cases all session keys can be recovered.

Depending on the implementation of the challenge/response mechanism on the smart cards, the vulnerability of old session keys may be eliminated for level 1 attacks. The challenge/response mechanism has to be based on a reversible function, such as DES rather than on a keyed one way function. The ANSI X9.9 MAC generation algorithm used in the IBM Multi Function Chipcards is an example. It essentially is DES run in CBC mode.

This algorithm has an 'inverse', that calculates a challenge from a message, a key and the corresponding Message authentication Code. In other words: let M be a message, K be a key, C be a challenge and R be the MAC of M under key K and challenge C (so $R = \text{MAC}_K(C, M)$) then $C = \text{MAC}_K^{-1}(M, R)$. With an inverse MAC one can take advantage of the fact that the

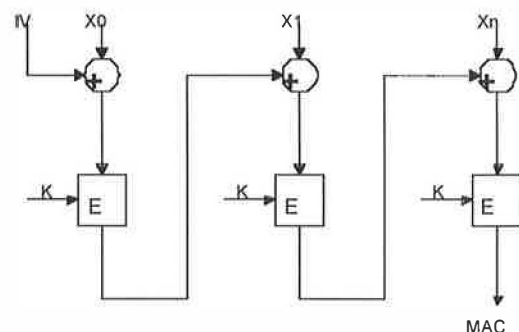


Figure 2: ANSI X9.9 MAC algorithm

TTP has full access to the keys in the identifiers, but users of the identifiers do not: a user can encrypt a short message by simply having the identifier perform a MAC calculation with the message used as challenge. The TTP decrypts the message by applying the inverse MAC. The holder of the identifier cannot, because the identifier restricts the application of the key it holds to the forward MAC calculation.

To strengthen the authentication protocol with reversible MAC's the following formulas have to be used for the notarization keys K_{A*} and K_{B*} :

$$K_{A*} = \text{MAC}_A(C_A, D_A) + \text{MAC}^{-1}_A(N_A, D_A) = \text{MAC}_A(C_A, D_A) \oplus R_A$$

$$K_{B*} = \text{MAC}_B(C_B, D_B) + \text{MAC}^{-1}_B(N_B, D_B) = \text{MAC}_B(C_B, D_B) \oplus R_B$$

where $N_A = \text{MAC}_A(R_A, D_A)$ and $N_B = \text{MAC}_B(R_B, D_B)$

Now the notarization keys are offset with the randomly chosen values R_A and R_B respectively. The TTP can decode these random values because they are encoded in the nonces N_A and N_B with a MAC calculation. Since a smart card does not offer a method to derive R_A or R_B from N_A and N_B an attacker cannot calculate the notarization keys and consequently the session key.

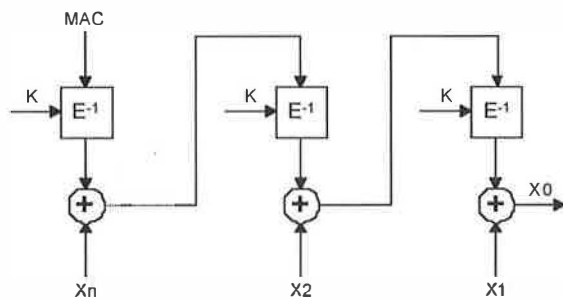


Figure 3: 'inverse' of X9.9 MAC

The TTP, on the other may derive the random values by applying the inverse MAC algorithm.

3. Prototype Implementation

In order to test the applicability and usability of the authentication protocol in actual WWW browser environments a prototype implementation has been built. Minimal goals for the implementation were Java 1.1 compatibility with support for at least one RS232 connected smart card reader and a commonly available smart card. For the latter, the ZeelandKaart and the Chipper, both incarnations of the IBM Multi Function Card were chosen.

The prototype implementation has been split into two parts: an implementation of the authentication and key transportation protocol and an implementation of a smart card access library. The intention was to be able

to plug in specific identifiers in the authentication protocol, including ones not based on smart cards.

Flexibility was an important design consideration. Adding support for new smart card interfaces or smart cards has to be easy. Also extending the protocol for extra security or new features must not break compatibility.

3.1. Smart Card Interfacing

At the start of this project no Java API for accessing smart cards was available so one was developed as part of the project. The Smart Card API is layered analogous to the ISO 7816 standards:

- Hardware layer: provides uniform access to different smart card interface devices.
- Datalink layer: implements the T=0, T=1, etc datalink protocols described in ISO 7816-3
- Transport layer: provides multiple concurrent smart card communication channels for datalink layers that support it (T=1).
- Application layer: implements the application commands standardized in ISO 7816-4 and prEN726-3

In order to maximize portability the bridge between native code and Java was put in the hardware layer. This meant the development of a native RS232 port access library that can be linked by the Java Runtime through the Java Native Interface (JNI). This library has been implemented for the Win32 and Linux platform with the JDK 1.0, JDK 1.1 and Netscape 3 & 4 Java Runtime Environments. Support for the NCD Explora network computer was added in pure Java.

On top of the RS232 layer drivers have been built for the Towitoko ChipDrive (see [Towitoko97]) and the DumbMouse smart card interface [BillSF96].

For the complete API and JavaDoc documentation, see [Bakker98].

3.2. Smart Card based Identifiers

As part of this investigation several smart cards have been examined for their applicability as identifiers in the authentication protocol: the Studenten Chip Kaart (SCK) 1995/96, the Zeeland Kaart, the ChipKnip, the Postbank Chipper and the Studenten Chip Kaart 1997/98 supplied to students at the Technical University of Delft.

Except for the ChipKnip, which is a Bull CP8 Transac CC 60 payment card, all cards are based on the IBM Multi Function Card OS that implements (a subset of) the ETSI En726 smart card application layer. (All cards offer an ISO7816-4 command set, the ChipKnip over datalink layer T=0 and the others over T=1).

The ChipKnip turned out not to be compatible with KLOMP since it provides authentication only within the *proof of debit* step of a payment transaction, not separately [BeaNet96]. The other cards provide either a 'protected read' or En726 'read stamped' method, which both include ANSI X9.9 MACs. With these methods improved KLOMP can be implemented. Experiments proved the MAC implementation on the Zeeland Kaart to be insecure however.

3.3. Password based Identifiers

For testing purposes also a password based challenge/response identifier has been built. It can be considered a software emulation of a smart card that implements the ISO7816 internal authenticate command. The password-based identifier simply asks the user for a user ID and password to perform MAC calculations with. In this case the cryptographic functions are processed in the semi trusted computing base instead of in a trusted computing base. Of course the consequence is that compromise of the semi trusted computing base fully voids the security of the identifier.

3.4. Protocol Implementation

A prototype of KLOMP using password-based identifiers has been written for the Java 1.1 platform. Using smart card based identifiers was not possible,

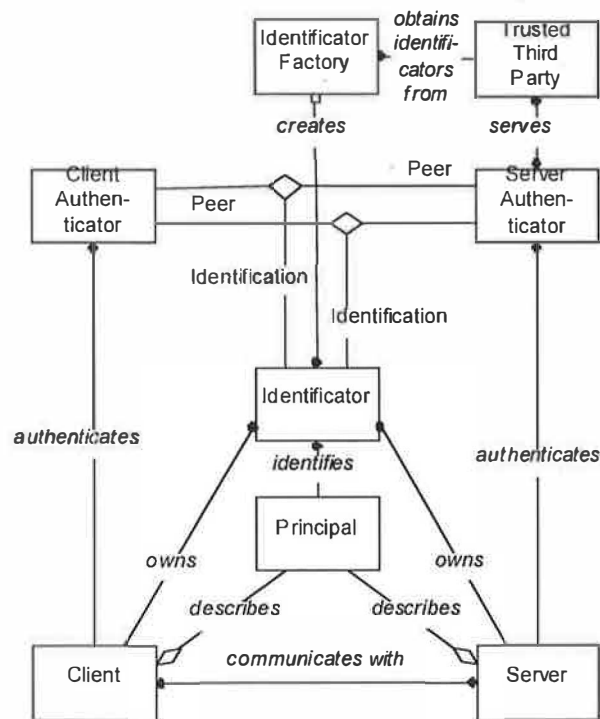


Figure 4: Class Diagram of the Protocol Implementation

because no smart cards were available of which the keys were known. Figure 4 shows a global class diagram of the prototype. At the TTP the identifier factory builds identifiers by looking up users and passwords in a PostgreSQL connected through JDBC. The ClientAuthenticator, the ServerAuthenticator and TrustedThirdParty communicate through Java Remote Method Invocation (RMI). Java 1.1 includes a class for generation of MD5 message digests, so this algorithm has been applied for all secure hashes. The prototype is available at [Bakker98-2].

3.5. Running the Prototype

Since the prototype protocol implementation is written in pure Java, so it should run with any Java 1.1 compliant runtime. (The prototype does connect to a pure Java UserID/Password identifier by default, therefore it does not need a native RS232 access library). The server also needs the Java Generic Library (version 3.0) and the TrustedThirdParty needs a JDBC driver, in this case one for PostgreSQL. A PostgreSQL database containing the user table has to be accessible for the TTP.

A small client has been written that repeatedly initiates the authentication protocol and times the duration for completion. The measurements of a typical run of this applet are shown below:

Pass	1	2	3	4
Time	5.542s	0.314s	0.337s	0.432s

Here the applet was running the protocol on a Pentium II 233 with Red Hat Linux 4.0 and JDK 1.1.1v2 generated the above log. The first run of the protocol takes clearly much longer than the other ones. This is due to the initialization of the RMI channels between the parties: it causes loading of many Java RMI classes into the runtime. Also the Java serialization mechanism has to perform elaborate hash calculations for every class serialized. These timings should not be considered an indication of the speed of the protocol in a typical application environment: the JDK 1.1.1 for Linux is much slower than the average Java implementation, for example it does not contain a Just In Time (JIT) compiler. The above test runs were performed with all three processes (client, server and TTP process) and the PostgreSQL database on the same machine. Several other test runs have been performed with these parts running on separate machines running different operating systems. Besides Linux the system was tested on Sparc Solaris 2.5.1 and WindowsNT 4.0. Other than speed no functional differences were observed with the different configurations. Also the PostgreSQL JDBC driver did not have any problems in accessing

the database remotely. All in all the binary code portability of the Java prototype implementation scored 100%.

3.6. Application of the Smart Card API

Separately from the protocol prototype, the smart card access code has been applied in a project for the Landelijk Instituut voor Sociale Verzekeringen (LISV). In this demonstration system people can register and login to the HTTP based application by entering their Chipper or ChipKnip card. A Java applet reads the account number and a password from the card and sends it to the web server for simple password based authentication. The applet takes less than half a second to read the card and send the login request. Furthermore the applet couples to (Netscape) JavaScript. At the registration page a Javascript routine uses this coupling to read name and address information from the Chipper card and fill the registration request form with it.

4. Conclusions

The ISO7816 standards for smart cards provide basic functionality for authentication through the internal authenticate command. Together with the storage of an ID this command allows us to build challenge/response identifiers. It has been demonstrated that with such identifiers it's possible to build an intrinsically secure three party authentication protocol (called KLOMP). Furthermore KLOMP imposes few requirements on the trusted party, e.g. no administration of sessions is needed for the authentication, no information has to be returned before both client and server have been authenticated. Also KLOMP separates authentication from privacy through strong encryption, so the system does not suffer from cryptography export regulations like those imposed by the US. The incorporation of encryption of short messages with 'inverse' MAC's strongly enhances the security of the protocol to a point where session keys are still secure even the smart card is stolen or otherwise abused. This gives hardware-based identifiers like smart cards a definite advantage over mechanisms that rely solely on software.

Currently the majority smart cards used in Holland are either a version of the IBM Multi Function Card (the "Chipper") or a ChipKnip. This is mainly a consequence of the massive scale at which the joined banks and the PostBank introduced the ChipKnip and the Chipper to the public. Though both are ISO7816 compliant cards, neither base their security on the internal authenticate command (even though the MFC at least implements it). The MFC does provide two other challenge/response based commands that can be

used as a substitution. The MFC therefore is compatible with the proposed authentication protocol. Unfortunately the ChipKnip cannot be adapted to the protocol because it lacks a transactionless authentication method.

The authentication protocol has been successfully implemented in Java. It has been tested on the Linux, Solaris and Windows32 platforms, demonstrating the cross platform capabilities of Java. The time measurements performed with the relatively slow Linux JDK 1.1.1 indicate that the protocol does not require an unacceptably long time to complete.

Bibliography

- [Bakker98] "Smart Card Access Library v1.1", Bastiaan Bakker 1998, <http://speeltuin.lifeline.nl/~bastiaan/smartcardapi.html>
- [Bakker98-2] "Klomp Prototype Implementation v1.0", Bastiaan Bakker 1998, <http://speeltuin.lifeline.nl/~bastiaan/klompproto.html>
- [BeaNet96] "ChipKnip Terminal Specifications", BeaNNet BV 1996
- [BillSF96] "Everything About Chipcards", Bill SF, magazine 't Klaphek, issue 2, 1996
- [Bird92] "Systematic Design of a Family of Attack-Resistant Authentication Protocols", R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, IBM 1992
- [Bird93] "The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution", R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, IBM 1993
- [ETSI93] "Requirements for IC cards and terminals for telecommunication use, part 3", prEN726-3 version 14, ETSI STC9, Valbonne Cedex 1993
- [Hoekstra97] "Design and implementation of the ISI-3 authentication protocol", Arjen Hoekstra, ISCIT 1997
- [ISO89] "Identification cards - Integrated circuit(s) card with contacts, part 3", ISO/IEC 7816-3, Geneva 1989
- [ISO92] "Identification cards - Integrated circuit(s) card with contacts, part 3, amendment 1", ISO/IEC 7816-3 Amendment 1, Geneva 1992
- [ISO93] "Identification cards - Integrated circuit(s) card with contacts, part 4", ISO/IEC 7816-4, Geneva 1993

- [IBM96] "The IBM MultiFunction Card Programmer's Reference v3.5", IBM Germany Development Laboratory, Boeblingen 1996. CONFIDENTIAL
- [JavaSoft97a] "Java 1.1 API Documentation", JavaSoft 1997
- [Schneier96] "Applied Cryptography, second edition", Bruce Schneier, John Wiley & Sons 1996, ISBN 0-481-11709-9
- [Towitoko97] "RS232 protocoll description for ChipDrive & KartenZwerg", Towitoko Electronics 1997

Notational conventions

$X // Y$	X concatenated with Y
$X \oplus Y$	X exclusive ORed with Y
K_X	Party X's master secret (= a long term key shared with the TTP)
K_{X_s}	Party X's key transforming key (= a derived key shared with the TTP for the duration of a session)
K_{XY}	Secret session key shared by X and Y
$E_X(M)$	Encryption of M under key K_X
$E_X^{-1}(M)$	Decryption of M under key K_X
$H(M_1, \dots, M_n)$	a secure hash of M_1, \dots, M_n
$MAC_X(M)$	Message Authentication Code for message M under key K_X
$MAC_X(C, M)$	Message Authentication Code for message M under key K_X with challenge C. The algorithm by which the challenge is inserted in the MAC is either unknown or undetermined.
$MAC_X^{-1}(R, M)$	Inverse MAC for message M under key K_X with response R: $MAC_X(C, M) = R \Leftrightarrow MAC_X^{-1}(R, M) = C$
N_X	a nonce generated by X
C_X	a challenge for I_X
I_X	The identifier owned by X
D_X	Data used by I_X to calculate MACs
M_X	a MAC calculated by I_X based upon challenge C_X
P_{XY}	Proof of authentication by X for verification at Y
T_{XY}	a ticket for X containing the encrypted session key K_{XY}
A	The client
B	The server
KDC	The Key Distribution Center (= the Trusted Third Party)
T	a timestamp

Abbreviations

ABK(t)	Three party authentication protocol in which a time stamp is used to substitute a nonce for party K.
CA	Certificate Authority
CBC	Cipher Block Chaining (mode)
CHV	Card Holder Verification (number)
ECB	Electronic Code Book (mode)
DES	Data Encryption Standard
DSS	Digital Signature Standard
ETSI	European Telecommunication Standards Institute
HTTP	Hyper Text Transfer Protocol
ICC	Integrated Circuit Card
ICV	Initial Chaining Vector
IEP	Intersector Electronic Purse
JDK	Java Development Kit
JVM	Java Virtual Machine
JNI	Java Native Interface
KDC	Key Distribution Center
KEK	Key Encrypting Key
KET	Key Expiration Time
KGK	Key Generating Key
KTK	Key Transforming Key
MD5	Message Digest number 5
MFC	Multi Function (Chip)Card
NC	Network Computer
PIN	Personal Identification Number
PTS	Payment Terminal System
RMI	Remote Method Invocation
SAM	Secure Application Module
SCM	Secure Cryptographic Module
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TTP	Trusted Third Party

Software license management with smart cards

Tuomas Aura

*Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O.Box 5400, FIN-02015 HUT, Finland
Tuomas.Aura@hut.fi*

Dieter Gollmann

*Microsoft Research
St. George House, 1 Guildhall St.
Cambridge, CB2 3NH, UK
diego@microsoft.com*

Abstract

This paper describes public-key protocols for binding software licenses to tamper-resistant smart cards, for transferring licenses between cards, and for purchasing them on-line. The protocols support software distribution both through retail stores and over the Internet. The user can transfer licenses from several cards onto a single card to avoid juggling between several cards in the reader. The protocols are based on signed delegation certificates that are mostly stored outside the smart card. A smart card reader and cards capable of public-key signatures are the only new hardware needed. The protocols are easy for the user and simple to implement and analyze. We prove the security of the transfer protocol.

1 Introduction

Unlicensed use of computer software has always been a major concern for the software industry. Lately, the piracy problem has been highlighted by the introduction of the Internet as a distribution channel [11], and the rise of content industry whose products are often collectively labeled as multimedia.

Most copy-protection and license management techniques have either proven ineffective or too restrictive for users to accept them. Thus, a majority of mass-market software products today are sold without any technological protection, which leaves marketing and legal battles as the only means for the software industry to defend itself. However, current advances in technology are opening new possibilities whose impact on license management should be assessed. First, with the popularity of smart cards, intelligent hardware tokens are becoming much more affordable. Second, computer networking makes two-way communication between the

customer and the software publisher more convenient.

This paper shows how these new technologies add a great degree of flexibility and ease of use to software license management and hardware-based copy protection. It is possible to use the new technique in combination both with conventional software sales through retail stores and with Internet commerce. The protocols proposed in this paper require public-key cryptography on the smart cards but otherwise they are extremely simple. The license information is in the form of signed certificates and can be managed mostly outside the smart cards.

The main threats that we address are multiple installations of software from a single-license distribution medium and production of counterfeit copies by professional pirates. These types of copying appear to have the greatest impact on the software publishers' revenues.

Another recent development, robust copyright-marking techniques such as watermarking [9], helps in resolving legal disputes over the ownership of data. However, it does not prevent copying of programs because the owner and copyright status are normally obvious from software products. A level of access control is needed to help the users make the right choice. It should be easier to buy than to copy. Also, copy-resistant physical tokens are needed to slow down the professional pirates whose aim is to mass-produce copies and market those as originals. We recognize that there are always ways to work around the protection mechanisms. What can be done is to increase the time to market for pirated copies and to ensure that pirated products cannot be sold as authentic to unsuspecting customers. If honest and security-conscious users are alarmed about tampered products, they are likely to buy authentic ones instead.

The rest of the paper is organized as follows. We begin with a short introduction to copy protection with tamper-resistant modules in Sec. 2. Sec. 3 gives an

overview of license transfer and Sec. 4 the protocol details. Sec. 5 continues with a protocol for on-line purchase of licenses. Techniques for strengthening the copy-protection are discussed in Sec. 6 and prevention of license theft in Sec. 7. Finally, we summarize the assumptions and advantages of the suggested protocols in Sec. 8 and list some possible extensions in Sec. 9. Sec. 10 concludes the paper. The Appendix contains a proof that the protocols cannot be subverted to copy licenses.

2 Copy protection with smart cards

The only theoretically secure copy-protection arrangement is to deliver the code in encrypted form and to decrypt and execute it inside a tamper-resistant processor [14, 15, 6]. In practice, such processors cannot be mandated and the code is exposed to insecure user equipment. Therefore, copy-protection is always to some extent security by obscurity.

In practical protection mechanisms based on a hardware token, a user license is embodied by a copy-resistant piece of hardware. The software or the operating system checks for the presence of the token and refuses to run without it.

A common type of token is a *dongle* that is inserted in a communications port on the workstation that is to run the software. If smart card readers become more common, a smart card is the obvious choice for a token. This is because the production cost of a single smart card is negligible compared to the cost of a software license. It would not be impossible to routinely distribute smart cards with all shrink-wrap software.

Robust mechanisms for checking the authenticity of the hardware token are based on a cryptographic key that is never stored or used outside the tamper-resistant token. The security of the mechanisms depends on two assumptions of technical intractability: it must be too expensive or time-consuming to reverse engineer the smart card in order to obtain the hidden secrets in it, and it must be equally difficult to modify the software to run without the card. Both of the assumptions present difficult problems of their own. This paper leaves them for others to solve. Tamper-resistant smart card technology is an active area of research [1, 8, 10], as is authenticated booting of software.

Unfortunately, dongles or smart cards are unpopular

with users. The main objection has been that the protection mechanisms for different software packages often interfere with each other. Even if the protection mechanism for each individual product is well designed, they might become unusable together. This is a major problem for smart cards since a single card must not be allowed to monopolize the card reader. In order to prove presence of a token for different software packages, one may have to repeatedly insert different cards into the reader, an annoying practice sometimes referred to as *smart card juggling*.

We will describe a solution for binding software licenses to smart cards and for transferring them from card to card in such a way that the juggling is eliminated.

3 License transfer with delegation certificates

In order to achieve flexibility and ease of use, our goal is to allow a single smart card to act as a token for arbitrarily many software packages. The licenses are distributed on cards that the customers get bundled with each software package. Normally each card holds only a single license. With a simple procedure, the licenses on one card can be transferred onto another card. After the transfer, the “empty” card may be discarded.

Every card has a unique public-private key pair. The private key is stored on the card and never revealed to the outside. At any time when the card is in the reader, it will respond to a challenge to prove that it, indeed, has the private key corresponding to the public key. This way, the software can check that the card associated with the license is present in the reader.

It is still necessary to bind a license to the public key. A convenient way to do this is to issue a certificate to the public key of the card. The certificate will be signed by the software publisher’s master key and it will be verified with a public key incorporated in the software or in the operating system. The certificate can be stored outside the card. In fact, the card never needs to know which licenses it is certified to have.

It is crucial that the publisher’s master public key and the procedure for checking the certificate and the presence of the card are embedded in the software in such a way that the key cannot be changed and the check cannot be disabled. In general, this is not an easy task. The protection can always be removed by reverse engineer-

ing the code. In practice, however, obfuscation of the checking procedure can significantly delay the reverse-engineering process and the production of marketable copies. Section 6 describes some measures that make the marketing of the modified software less attractive after the protections have been removed.

We will now outline the mechanism for transferring licenses from one card to another. Once the license is bound to a public key, it can be given to other keys by *delegation*. The key having a license simply signs a certificate stating its willingness to give the same rights also to the key of another card. This kind of certificate with which one key delegates access rights to another one is called a *delegation certificate*. The signing is done with public-key cryptography. It is possible to use standard certificate formats and techniques [5, 2].

Unfortunately, this simple procedure is not quite enough; it would result in duplication of the license. After handing over the license, the first card must cease to function as a token. Furthermore, it must never sign another delegation certificate (at least not to delegate the same license). The simplest way to ensure this is to erase the private key from the delegating card. Erasing the key means that we must always transfer all licenses together to the same card and then discard the original card.

Thus, license transfer comprises two steps:

1. delegating the license to another card
2. disabling the delegating card as a token.

In principle, a license can be transferred an unlimited number of times. Every transfer adds a new delegation certificate to a chain that passes a growing collection of licenses from card key to card key. When the right to use a certain software package is verified, there must be a complete chain of certificates starting from a license certificates signed by the publisher's master key and ending with the public key of the card that is currently in the smart card reader. In practice, the number of transfers for a single license will be very small (usually one). The licenses cannot be transferred onto previously disabled cards and every transfer accumulates all the licenses on two cards onto a single one.

The cards need to have only two basic functions: proving the possession of a private key by responding to a challenge and signing a delegation certificate after which the card disables itself. Further management of the certificates is done outside the smart card. Sec. 4 details the transfer protocol.

The idea of transferring licenses from one smart card to another bears resemblance to the transferable digital cash of Pagnia and Jansen [12]. From the result of Chaum and Pedersen [4], we can infer that the growth of the license data with each transfer is inevitable. In our scheme, another delegation certificate will be appended to the license in each step. It is important, however, to note that the growing collection of delegation certificates is not stored in or processed on the smart cards. The smart cards act as tamper-resistant observers that guard against copying of licenses. This is equivalent to double spending prevention for digital money [3].

4 Protocol for license verification and transfer

Each card has a unique signature key pair. The public part of the card key (CK) can be read from the card at any time while the private key is never exported outside the card. In addition to the keys, the card stores a *card certificate* signed by the software-publisher master key (PMK). The card certificate states that CK is the public key of an authentic license card. The certificate can be freely read from the card. Anyone knowing the public PMK can verify the certificate on the card and conclude that this is an authentic license card approved by the software publisher. The PMK and the card certificate will be used to ensure that licenses are transferred only to smart cards that reliably protect them from copying. Every card has to store the authentic public PMK for verifying card certificates. A card certificate is of the form

$$S_{PMK}(CK, \text{"is a license card key with production date", date})$$

(The notation $S_K(M)$ means the message M signed with the key K . Our view of the signature function is ideal. Implementations should follow accepted standards such as PKCS [7].)

The certificate contains the production date or serial number of the card. The licenses can only be transferred from older to newer cards. This ensures that cryptanalysis of old card keys or cracking the defenses of old tamper-resistant cards cannot be used for copying new software products. It should be noted that the card certificates and the dates on them originate from the software publisher or trusted card manufacturers, the date

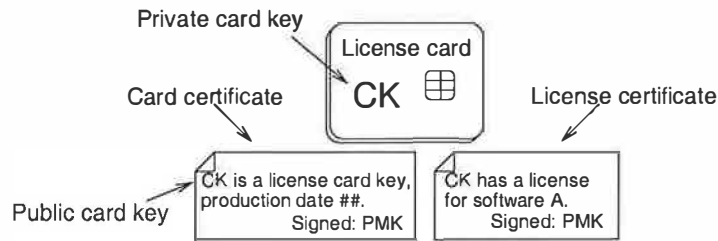


Figure 1: A basic license card has a key pair and two certificates.

stamps are compared against each other, and the comparison is done on the tamper-resistant license cards. No clocks are needed on the smart cards and clocks in the user equipment are not relied on. Therefore, the comparison of dates is reliable. As a secondary protection, the software checking the presence of the token should also ensure that the date on the card certificate is not older than the software itself.

Normal cards originally hold only one license as they are sold in retail stores with software packages. The license is a certificate signed by the *PMK* that binds the right to use a certain software package to the *CK*. The *license certificate* is of the form

$S_{PMK}(CK, \text{"has a license for"}, \text{software}).$

Although it is not necessary to store the license information on the card, it is convenient to distribute licenses on the cards that come bundled with the software distribution media. This way, the software itself can be on identical media, e.g. printed CD-ROMs. For efficiency, it is best to read the card and license certificates from the card into the workstation only once when the software is installed and never refer to them on the card again. It is possible to ship several license certificates with a single card (e.g. stored on a floppy disk). This is practical when the publisher ships products directly to the users or when workstation manufacturers pre-install a standard set of software.

In addition to carrying the certificates, the card can perform two main functions: proof of identity and license transfer. The former means proving the possession of the private *CK*. The card does this simply by signing a challenge.

Protocol 1 (proof of identity):

1. Workstation \rightarrow Card :
"License card challenge", N
2. Card \rightarrow Workstation :
 $S_{CK}(\text{"License card response"}, N)$

The checking software first needs the public card key *CK* and the card certificate. Then, it can send a challenge to the card and verify that the card is authentic. To decide if the card has a certain license, the software follows the delegation certificates to find a chain of delegation from *PMK* to *CK*. These certificates are stored outside the card and can be written into a file or onto a floppy disk for keeping with the card.

The delegation certificates are created in the second main function of the card, the license transfer. All licenses on the card are always transferred at the same time. After the transfer, the original card can be thrown away.

The transfer protocol is very simple. Licenses on two cards will be combined onto one of them. The source card (the one to transfer from) must be the older card and the destination card (the one to transfer to) the newer one. Before the transfer, the workstation obtains the card certificate of the destination card. After that, the protocol is between the workstation and the source card only. The destination card is not involved in the communication. The delegation certificate produced in the transfer will be stored in the workstation and it will later be used together with the destination card. However, the destination card does not need to know anything about the transfer and the certificate is never saved onto the card.

Protocol 2 (license transfer):

1. Workstation \rightarrow Source card :
 "Please transfer to", CK' ,
 $S_{PMK}(CK')$, "is a license card key
 with production date", $date$)
2. The source card signs a certificate and
 erases its private key CK .
3. Source card \rightarrow Workstation :
 S_{CK} ("I give all my licenses to", CK')

In the first step of the protocol, before delegating the license to the public key CK' , the source card checks that the key belongs to an authentic license card. It therefore needs the card certificate for CK' . It also compares the date on the card certificate to its own production date to see that the destination card is the same age or newer.

In step 2, the card signs a *delegation certificate* for CK' . The certificate is of the form S_{CK} ("I give all my licenses to", CK'). After signing the certificate, the card permanently erases its own private key CK from its memory. After erasing the key, the card is not anymore able to perform Protocol 1, i.e. the proof of identity. This means that it is disabled as a license token.

Having created the delegation certificate, the card returns it to the requesting workstation in Step 3 of the transfer protocol. After the transfer, the source card is useless and it can be thrown away. In order to protect against loss of certificates, the card certificate, the license certificate and the new delegation certificate are still stored on the otherwise disabled card and can be reread an unlimited number of times.

A crucial point for the smart card implementation is that signing the delegation certificate and erasing key private key must be an atomic operation. If the operation is interrupted, for example, by cutting power from the card, the card must either complete the signing and erasure immediately after power-up, or it must return to the original state where the delegation certificate does not exist. Moreover, the production and storage of the delegation certificate on the card must be reliable because the signing cannot be repeated after the private key has been erased.

In summary, the protocol performs the two steps that make a complete transfer: delegation and disabling the old card as a token. In the workstation, the new delegation certificate will be combined with the ones both

cards previously had. All these certificates are needed for use with the destination card (Fig. 4).

5 On-line software distribution

Although we cannot assume all customers or all workstations to have Internet connections, an increasing number of customers is willing to purchase software on-line. Two-way communication between the user workstation and the software publisher opens new possibilities for license management. It is necessary for the same license management system to support both traditional shrink-wrap software sales and on-line commerce.

When licenses are sold on-line, they can be personalized for each customer. The key to controlling the distribution of the licenses is to bind each license to exactly one user workstation at a time. There are plans for incorporating unique identifiers into the microprocessors in personal computers for this purpose. Because of privacy concerns, it is not clear whether such identifiers will ever be implemented by all vendors. Some copy protection products compute a fingerprint of the hardware and software configuration to identify the workstation [13]. Unfortunately, this may cause invalidation of the license when parts of the system are updated.

In our system, the card key of a smart card is a unique identifier to which the licenses are bound. The same smart cards that are sold in retail stores can be used for on-line purchases. Instead of getting the licenses with the card, the customer buys license certificates for his card from the on-line store. If the particular workstation already has a license management card, the license certificate will be issued to the public card key of that card. Most customers have recent cards e.g. from purchasing the operating system and, since the price of the smart card itself is low, empty cards without a license can be distributed free of charge.

Protocol 3 (on-line purchase):

1. Customer \rightarrow Publisher :
 "I buy a license for", CK ,
 $S_{PMK}(CK)$, "is a license card key
 produced on ", $date$)
2. Publisher \rightarrow Customer :
 $S_{PMK}(CK)$, "has a license for", *software*)

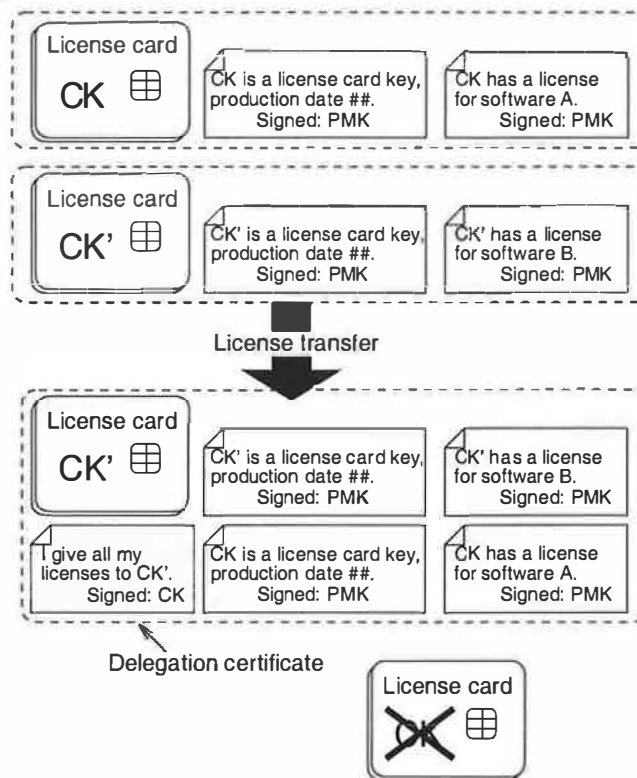


Figure 2: License transfer = signing a delegation certificate + disabling the source card. The certificates are stored outside the cards.

The on-line store needs to see the card certificate to check that the public key *CK* belongs to an authentic license card.

The only limitation for this type of on-line sales is that licenses should not be sold to cards that are too old because their keys might have already been recovered by pirates. (If the pirate knows the private key of an authentic license card, he can purchase one license on-line and delegate it to any number of cards.) If the card is older than some threshold time, the customer needs to obtain a new card and move licenses from his old card onto it before buying new software on-line. The threshold time for rejecting old cards can be adjusted for new products according to the experiences from earlier releases.

Actually, it is not important how the software itself is distributed: on-line or on a CD-ROM or on some other medium. The distribution of licenses can be completely independent of the software distribution.

On-line services open new possibilities for strengthening the security of the system. If the product includes parts or services, such as updates, that are delivered over the

Internet, the servers can check for the license before providing the service. The on-line server will send the user workstation a challenge. The response from the smart card is sent to the on-line service along with the public key of the smart card, the license certificate or a chain of certificates from *PMK* to *CK*, and the card certificates for all card keys in the delegation chain. The server can store fingerprints of the keys in the chain and refuse to repeatedly provide the same service for the same keys. Similarly, the same license should not be sold twice to the same card because it may indicate that the card is a clone. This improves the strength of the copy protection in case a pirate is able to recover the private key of a single card. Users of the pirated licenses will be refused on-line service.

It is for the purpose of bookkeeping at the on-line servers that we retain all the card certificates in the license transfer. If on-line services are not available or the on-line server has a database of all valid card keys, it is not necessary to store the card certificates of the disabled cards after the transfer. It suffices to keep the one belonging to the active card.

6 Enhancing the copy protection

Copy protection is never perfect. Therefore, we will consider ways of strengthening the protection. The effectiveness of these techniques depends on the nature of the product and the environment where it is mainly used.

The two most dangerous attacks against the copy protection in our license management scheme are recovering of the private key from the smart card and modifying of the software to bypass the checking for the token.

When on-line updates or other Internet services are an essential part of the product, the problem of recovered private keys is alleviated by having the servers remember the keys for which the service has already been provided (see Sec. 5). Professional pirates cannot produce fully functional copies even if they are able to crack the protections of a single card because users of the illegal copies will not be able to access the on-line services. This works because the cards have unique keys instead of one shared secret. Continuing the analogy to digital cash, the database of served card keys resembles double-spending detection by banks that keep track of spent coins.

Another way of discouraging the purchase of pirated copies is to have the users authenticate the software. The person installing or using a pirated software package should get a warning about potentially dangerous, unauthentic code. The warning can be implemented by signing the code with the publisher master key *PMK* or with another key held by the publisher. The public verification keys can be distributed on-line or with the operating system. If the pirates modify the software in order to remove the check for the license, the pirated copies inevitably fail the test for correct signatures. This kind of integrity check is beneficial even if copy protection is not an issue: software distributed over the Internet should be authenticated in any case.

Implementation of the integrity warning messages requires co-operation with the operating system or with a generic installation program. The warnings can naturally be avoided by modifying also these support programs. However, most business users of software are probably unwilling to tweak their operating system according to the pirate's instructions. Also, the embedded operating systems in special-purpose devices such as game consoles and multimedia terminals often cannot be modified by the user.

Instead of completely disabling the check for the to-

ken, pirates may try to modify the smart card reader or its driver software in such a way that several workstations can share one reader. The challenges and responses could be transferred over the network between a single reader and a large number of verifiers. To prevent such modifications, the checking software should have direct access to the smart card reader hardware so that it can trust the responses to be from a local source. Sometimes, especially if the operating system consists of replaceable modules or layers, it may be impossible to prevent tapping between the card and the verifier. Even then, the attack is only possible if the modification to the operating system is available and if the user organization is willing to install the patches on all workstations. Other possible defenses include binding the licenses to workstation identities and limiting the number and frequency of answered challenges per license. Such measures, however, imply unique processor identifiers, on-card timers, counters, and much more complex protocols that are beyond the scope of this paper.

There is one effective technique for checking the presence of the token which requires some adjustment for our public-key protocols. That is, the software on the distribution media can be encrypted and the license token should contain the key for decrypting it. If the software is never stored outside the computer memory in decrypted form, it cannot be loaded without the token. (Naturally, this is just a way of obscuring the check for the tokens. The program in the insecure computer memory can be read and saved with special tools and skillful reverse engineering.) Dongles sometimes carry a secret key for decrypting the code [13]. If we want to use this technique in our license transfer scheme, we have to pass the decryption keys to the destination card and erase them from the source card as a part of the license transfer. The keys can be transferred by encrypting them with the public key of the receiving card.

7 Preventing license theft

New technology often creates new types of vulnerabilities that are beyond our prior experience. When the software licenses are bound to small, tangible objects, a new threat emerges: theft of licenses. And what is most disconcerting, the transfer protocol could be misused to steal licenses electronically over the network. Luckily, theft can be prevented with simple password protection.

The physical theft of license cards may be a problem anywhere where untrusted persons have physical access

to the workstations. The standard protection against the theft of smart cards is that the card requires the user to enter a password after it is inserted into the reader. The card refuses to work unless activated with the correct password. This effectively prevents the use of stolen license cards. The password can be distributed on paper with the card. For convenience, the users should be able to disable the password feature in environments where theft is not a major threat.

Even with the password protection, there is still the danger that someone removes the card not for his own profit but to cause damage to the owner. Vandalism is a problem for public-access computers in places like universities and libraries. The card could be protected by enclosing the card reader inside the workstation casing or by using lockable special-purpose readers.

A more interesting scenario is that the thief transfers the license onto his own card. A hacker could even break into the computer from the network and invoke the transfer procedure without having physical access and without exposing himself to much danger of being identified. Again, a separate one-time password should be required by the card before the transfer. Since the transfer is activated only once for each card, passive sniffing for the passwords does not benefit an attacker. In theory, the hacker could take over the transfer process after the user has entered the password and replace the destination card certificate with his own. Therefore, license transfers should be done on a trusted workstation, preferably off-line. An alternative protection that prevents attacks from the network is a physical write-protect switch on the card that must be shifted to allow the transfer.

8 Evaluation

The main goal in the development of our license management scheme was to make the use of hardware tokens user-friendly. In particular, we have solved the problem of smart-card juggling. Although the user must insert a smart card into the reader, it is not any more necessary to periodically switch between cards. Licenses of several software packages are transferred onto a single card.

The license transfer is an extremely simple procedure for the user. He inserts the two cards into the smart card reader, the newer card first. (If the order is wrong, he is asked to reinsert the first card.) He may be asked to provide a floppy disk for backing up the certificates.

The system requires the user to have a smart card reader on every workstation and the license card must be in the reader for most of the time. Beyond the card reader, no changes to existing hardware (e.g. Internet connection, secure processors or hardware identity numbers) are needed. After the initial investment in the readers, the marginal cost of protecting each new product is small. Since the certificates are stored and handled mostly outside the cards, the storage and computational capacity needed in the smart cards is bounded.

We have seamlessly integrated shrink-wrap and on-line sales of software licenses. The license management does not require any changes to existing software distribution channels. In particular, incorporating a smart card into shrink-wrap software packages does not increase the workload at retail stores or require users to have network connections.

The security of the system relies on two non-trivial assumptions. First, the smart card must be tamper resistant in the sense that the private key cannot be recovered from the card. Recovering the key of even one card makes it possible for a professional pirate to sell counterfeit licenses. With on-line distribution of licenses, the pirate must crack a new card when the previously cracked card becomes so old that the on-line store refuses to sell new licenses to it. If software is not sold on-line, the pirate must recover a new key for each new software product or version. It depends on the state of the tamper-resistant smart card technology how long it takes to analyze a card. Service can be denied to those users of pirated software who try to utilize on-line updates and services associated with the products.

Second, the checking for the token in the software package must be obscured in such a way that the check cannot be disabled. Since the public key of the software publisher (*PMK*) is used for the checking, one should not be able to change this key in the code. Like tamper-resistant cards, obscured software can be analyzed with time and resources. In this case, it is probably the easier line of attack. We suggest discouraging the use of modified software by issuing warnings to the user.

When these basic assumptions are satisfied, the license transfer protocol itself is fairly robust. The transfer process cannot be interrupted to prevent erasure of the old license because the private key is erased as soon as the delegation certificate has been signed. We state formally the claim that the protocols do not allow copying of licenses (see the Appendix for the proof):

Proposition 1: The number of keys with valid licenses is at most equal to the number license certificates signed by *PMK*.

Moreover, licenses are not easily lost because the delegation certificates can be reread from the card at any later time and the certificates are backed up on the workstation hard disks or on floppy disks.

In summary, the license management system prevents multiple use of one license and it increases significantly the work of professional pirates. Although the user must keep the license card in the smart card reader, it is much more convenient than having separate tokens for each product.

9 Protocol extensions

Our license management protocol can be extended in several ways to increase its flexibility for users.

Although the protocol is fairly robust against accidental loss of licenses, there should be some off-line recovery mechanisms in case the license card is damaged or the delegation certificates are lost. The software producer can be generous with replacing cards and lost licenses. If a log is kept of the customers who receive a replacement certificate or smart card, the number of customers willing to cheat to get one extra copy of the product is likely to be small.

Although we have designed the license transfer protocol with local transfer in mind, the protocol itself has no restriction for remote transfer over a network. If this is implemented, the customer can transfer licenses over large distances without waiting to get the physical license card in mail. The remote transfer cannot be abused so that several remote users would pool to share a license (with at most one user at a time) because a new smart card is needed for each transfer.

In order to have only a single card per workstation, software publishers must co-operate. All card must use the same protocol and meet the same standard of tamper-resistance. Fortunately, it is not necessary to have all publishers share the master key *PMK*. Instead, the products of each publisher can check for a delegation chain starting from its own master key. These publisher keys, however, cannot be used for signing the card certificates because the safety of the cards affects all pub-

lishers. For this purpose, another layer of delegation can be added: a trusted agency holding a master key that will certify card manufacturers' master keys. The manufacturer will include its certificate on the card and sign the card key *CK* with its own key. This allows accreditation of new manufacturers at any time.

Naturally, the use of the delegation certificates for license management is not restricted to smart cards. The same ideas could be used with any intelligent hardware tokens as long as the tokens are capable of processing public-key signatures and their cost is low enough so that they can be discarded. Different physical implementations of the tokens can be mixed as long as they follow the same protocols. Non-discardable physical tokens such as dongles and chips embedded in the computer hardware work well but licenses should be only transferred onto them, not from them. One possibility is to have one embedded token per workstation and to use smart cards only for license distribution.

Finally, an alternative to having a card for each workstation is to let each user carry a personal license card. That naturally leads to using the card as a general-purpose identifier for the user. Our protocols are equally well suited for many other purposes such as maintaining personal key rings for smart-card locks. However, such applications are beyond the scope of this paper.

10 Conclusion

We have described protocols for binding software licenses to tamper-resistant smart cards, for transferring them between cards and for buying licenses on-line. There must be smart card readers at the workstations but no network connection or other changes to existing hardware are needed. The protocols support software distribution both through retail stores and over the Internet. The user can transfer licenses from several cards onto a single card so that juggling between several card in the reader is eliminated. The transfer protocol is easy and intuitive for the user. The smart cards must be able to process public-key signatures. In other respects, the protocols are simple both to implement and to analyze. Most of the data involved is stored outside the smart card. The protocols may also have applications in other system where smart cards are used for storing credentials.

11 Acknowledgments

Tuomas Aura's work was funded by Helsinki Graduate School for Computer Science and Engineering (HeCSE) and Academy of Finland and it was done mostly at UC Davis Computer Security Laboratory.

References

- [1] Ross Anderson and Markus Kuhn. Tamper resistance — a cautionary note. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 1–11. USENIX Association, November 1996.
- [2] Tuomas Aura. Distributed access-rights management with delegation certificates. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, LNCS. Springer, 1999.
- [3] Stefan Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology - Proceedings of CRYPTO '93*, volume 773 of LNCS, pages 302–318, Santa Barbara, 1993. Springer-Verlag.
- [4] David Chaum and Torben Pryds Pedersen. Transferred cash grows in size. In *Advances in Cryptology - Proceedings of EUROCRYPT '92*, volume 658 of LNCS, pages 390–407. Springer-Verlag, May 1992.
- [5] Carl M. Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. Simple public key certificate. Internet draft, IETF SPKI Working Group, March 1998.
- [6] Amir Herzberg and Shlomit S. Pinter. Public protection of software. *ACM Transactions on Computer Systems*, 5(4):371–393, November 1987.
- [7] Burt Kaliski and Jessica Staddon. PKCS #1: RSA cryptography specifications, version 2.0. Internet draft, IETF Network Working Group, September 1998.
- [8] Davis P. Maher. Fault induction attacks, tamper resistance, and hostile reverse engineering in perspective. In *Proc. 1st International Conference on Financial Cryptography FC '97*, volume 1318 of LNCS, pages 109–121, Anguilla, British West Indies, February 1997. Springer Verlag.
- [9] Nasir Menon and Ping Wah Wong. Protecting digital media content. *Communication of ACM*, 41(7):35–43, July 1998.
- [10] Mondex. Mondex home page, 1998. URL: <http://www.mondex.com>.
- [11] Information Technology Association of America. Intellectual property protection in cyberspace: towards a new consensus. ITAA discussion paper, 1998.
- [12] Hans-Henning Pagnia and Ralph Jansen. Towards multiple-payment schemes for digital money. In *Proc. 1st International Conference on Financial Cryptography FC '97*, volume 1318 of LNCS, pages 203–215, Anguilla, British West Indies, February 1997. Springer Verlag.
- [13] John Phipps. Physical protection devices. In Derrick Grover, editor, *The protection of Computer Software - its technology and applications*, British Computer Society (BCS) Monographs in Informatics, chapter 3. Cambridge University Press, 2nd edition, 1992.
- [14] George B. Purdy, Gustavus J. Simmons, and James A. Studier. A software protection scheme. In *Proc. 1982 Symposium on Security and Privacy*, pages 99–103, Oakland, California, April 1982. IEEE Computer Society Press.
- [15] Steve R. White. ABYSS: A trusted architecture for software protection. In *Proc. 1987 IEEE Symposium on Security and Privacy*, pages 38–51, Oakland, California, April 1987. IEEE Computer Society Press.

Appendix (proof of protocol security)

We consider the licenses for a single software product.

Definition 1: A key CK has a valid license if the private part of the key CK is unerased, and there is a card certificate signed by PMK and issued to the public part of CK , and a *certificate chain*:

a license certificate signed by PMK to CK_0 ,
 a delegation certificate signed by CK_0 to CK_1 ,
 a delegation certificate signed by CK_1 to CK_2 ,
 ⋮
 a delegation certificate signed by CK_{k-1} to CK_k

such that $CK_k = CK$. \square

This forms a valid license because the verifier checks for these conditions before allowing the use of the software. It is possible that $k = 0$, i.e. there are no delegation certificates. We ignore the check for the other card certificates (of $CK_0 \dots CK_{k-1}$) and for the production dates because they have effect only if some other assumption is broken.

Assumption 1: *PMK* only issues license and card certificates to authentic card keys. An authentic card key only issues delegation certificates to keys with a card certificate. \square

Assumption 2: For every authentic card key CK , exactly one of the following holds:

1. CK has signed no delegation certificates.
2. CK has signed exactly one delegation certificate and the private key CK has been erased. \square

The second assumption follows from the policy of erasing the private key immediately after signing a delegation certificate.

Proposition 1: The number of keys with valid licenses is at most equal to the number license certificates signed by *PMK*. \square

Proof: The subjects of license certificates (CK_0) are always authentic card keys. An authentic card key only delegates to a key with card certificate and such keys are authentic card keys (Ass. 1). Consequently, all keys in the chains starting from license certificates are authentic card keys. The authentic card keys delegate to at most one other key and a key that has delegated is itself erased (Ass. 2). Thus, the certificate chains starting from license certificates do not branch and only the last key in a chain can be unerased.

If there would be more keys with valid licenses than license certificates, there should be some two valid licenses whose corresponding certificate chains (Def. 1) begin with the same license certificate but end in two different unerased keys. However, this is not possible since the chains do not branch and only the maximal-length chains end in unerased keys.

Beyond Cryptographic Conditional Access

David M. Goldschlag

David W. Kravitz

Divx

570 Herndon Parkway

Herndon, VA 20170, USA

Abstract

Conditional access (CA) systems manage chargeable content (e.g., movies). Traditional CA systems use a smartcard as a cryptographic component that decrypts broadcast content for authorized recipients. Since that approach protects content by protecting cryptographic keys, it has two inherent weaknesses: It relies on the smartcard to protect universal secrets (i.e., the broadcast keys); and it cannot protect content from redistribution. This paper describes a non-cryptographic conditional access system, where instead of protecting content directly, the content's identity is inserted as a watermark in the content and the CA smartcard is used as a licensing authority to authorize the display device to display watermarked content. This approach places a lower security burden on individual smartcards, and protects against the use of redistributed content.

Keywords: Authentication, authorization, conditional access, copyright protection, licensing, smart cards, trust management, watermarking.

1. Introduction

Conditional access (CA) systems control access to chargeable content. This content includes both data and entertainment products such as movies or music. The content may be delivered in many ways, including broadcast from satellite or a local broadcaster, transmitted over cable or the Internet, or delivered by fixed media, such as a DVD. Billing policies also vary greatly: content may be sold as part of a subscription such as premium movie channels, by the unit like pay-per-view movies, or incrementally as in a stock ticker that is charged by access time.

Traditional CA systems protect chargeable content cryptographically. The primary function of a CA sys-

tem is to report content access to a billing system, and to prevent unreported access to content. To force content to be accessed through the CA system, content may be encrypted using cryptographic keys known only by the CA system. In that model, encrypted content is broadcast by a headend system and accessed through a CA smartcard on the user's set-top-box (STB, e.g., a TV or cable tuner). Accesses are reported to a backend billing system. The distribution of content keys is managed by a key management system (KMS) that lets CA smartcards learn the keys needed to decrypt content.

Cryptographic CA systems require the CA smartcard to protect content keys. Since efficient broadcasting requires universal shared secrets (i.e., there is a single broadcast data stream for any content), each smartcard must be trusted to protect universal secrets. This is a severe security burden for the smartcard. The pirate may use significant resources to compromise a single smartcard, and learn the keys necessary to decrypt content. Those keys are useful to all recipients of the content. Furthermore, CA systems are attempting to solve an inherent paradox: How is access to content enabled yet still controlled? The STB must have access to the unencrypted content. The pirate may attack his STB, obtain the content, and redistribute it. This threat has typically been called copy protection, and is not a CA threat, as long as redistribution is expensive.

We are concerned about CA in an environment where redistribution and storage of content and keys is cheap. Although this is not yet true for high quality video, it will be true, and it is appropriate to design systems that are resistant to this threat. Notice that display devices will always have to display free video, at least from camcorders. So the pirate can redistribute a pirated movie as a home video.

Our proposed architecture uses the CA smartcard as a licensing authority, instead of a decryption device. When granting authorization, a CA device may also log content access, so the access may be billed for. If dis-

play devices required authorization to display certain content, and would refuse to display the content in the absence of appropriate authorization, content would be secure independently of how it was delivered to the display device. The assumption here is that display devices like monitors are expensive and difficult to manufacture and service, and that pirates will not be able to compete in that market.

In this architecture, instead of protecting against the copying and redistribution of the content, we prevent unauthorized access to the content. This architecture places the security where it belongs, at the customer of the pirate, who will be unwilling to spend significant resources to defeat the system. In contrast, in cryptographic copy protection, the pirate can leverage off of his investment, without requiring further investment by the consumer.

This paper is organized in the following way: Section 2 presents an overview of traditional CA systems. Section 3 presents the non-cryptographic CA architecture. Section 4 describes related work. Section 5 presents some concluding remarks.

2. Cryptographic CA Overview

A model of a CA system is illustrated in Figure 1. Notice that content keys need not be delivered with the content, although they may be, and that the backend billing system may be independent of the headend too.

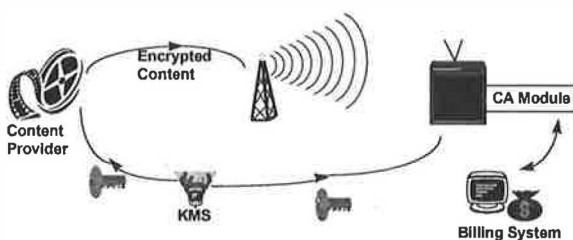


Figure 1: Model of CA Infrastructure

The billing system has two security roles: one is to instruct the key management system which keys should be sent to individual CA smartcards, and the other is to upload logs of key use from CA smartcards. These two functions correspond to advance billing for subscription services, and credit-type billing for pay-per-view serv-

ices (i.e., where services are used before they are paid for).

The CA smartcard, which is present in every user's house, must protect cryptographic keys that are universally useful. That is, the keys that a CA smartcard uses to decrypt content are equally useful to other recipients of the broadcast. If those keys are compromised and distributed, other recipients of the broadcast may use the keys to access content for free. In particular, since there is more content than keys (e.g., content takes up more bandwidth than keys), a pirate can leverage off of the original broadcast, only redistribute keys, and make money at the broadcaster's expense.

In cryptographic CA systems, the CA smartcard is a hardware device that protects keys. The security of this hardware decreases over time, because its design flaws become known, and physical attacks against it become cheaper. Furthermore, the pirate may be able to amortize the cost of attacking a single smartcard over the sale of many counterfeit smartcard. This suggests that the CA smartcard be renewable, so the security of the CA infrastructure may evolve over time. The challenge is to design an infrastructure that admits increasing security at low cost over a long lifetime.

3. Non-Cryptographic Conditional Access

If content redistribution is easy, there is no real point in protecting content up front, by encryption, for example. Imagine that bandwidth was very cheap: HBO may have a single customer, a pirate, who redistributes content to everyone else. Service providers cannot sustain businesses in such an environment. In this section, we describe a possible direction for protecting content in such a threat environment.

Notice that securing the input ports in the playback (display) device is not an adequate solution, since customers must always be able to play home-generated content (like output from a camcorder) and the pirate could redistribute content as if it were generated by a camcorder. We must prevent chargeable content from masquerading as free content. Even if in the future, camcorders would mark their content in such a way that chargeable content would not carry the free-content markings, this alone would not address the playing of legacy content generated by existing camcorders be-

cause a pirate could redistribute new chargeable content in the old home-generated content (unmarked) form.

The current reality of smartcard technology forces us to acknowledge that CA smartcards are not completely impervious to compromise. This together with the fact that at some point the plaintext content data is routed through the non-renewable playback device, forces us to focus on preventing successful use of the pirated content by potential customers of the pirate, rather than on preventing acquisition of the pirated content in the first place if we treat the content redistribution problem as a surmountable impediment. We want to take advantage of the fact that even if the pirate and his customers want to consider the use of pirate-provided non-compliant playback devices, the sophisticated monitor technology in display devices imposes significant barriers to entry for small- to mid- scale piracy operations.

Assume that the analog content itself can be watermarked in some robust way at authoring time, so legitimate playback devices will detect the watermark. Assume furthermore that the pirate cannot remove this watermark without significantly degrading the quality of the viewing experience. Watermark removal is usually done by corrupting the content, or by comparing two instances of the content with different watermarks. This latter attack need not be enabled in this approach, since all instances of the content will have the same watermark (i.e., watermarking is done here for playback control purposes, and not for tracing the source of unauthorized distribution). Unlike "fingerprinting," in this approach the content is watermarked at authoring time.

Assume that each legitimate playback device (i.e., TV monitor) is bound to a limited set of CA smartcards, so that it will refuse to accept critical commands from any smartcard which cannot authenticate these commands as being generated by a smartcard in the distinguished set. A playback device may be bound to a smartcard by receiving a binding certificate signed by a recognized binding authority that binds the playback device to the smartcard. The binding certificate gives the playback device permission to trust responses from the specified smartcard. The playback device need not be authenticated to the smartcard: That is, the smartcard will sign authorization requests from any playback device.

Notice that this binding between a playback device and a smartcard need not rely on the playback device having an externally discernible identity. For example, the playback device may issue a randomly generated bit-string, where the concatenation of this bit-string with

the public key of a smartcard with which the playback device is allowed to communicate, is signed by the recognized binding authority.

To play watermarked content, each legitimate playback device requires authorization from the CA smartcard each time it encounters an embedded watermark signal which differs from the one just previous. At the onset of attempted content play, the first embedded watermark signal always results in an authorization request. If this compression of challenges based on limiting authentication requests to deltas in the embedded signals proves to be ineffective when processing content because changes occur too frequently, either the authorization device or the display device may go into alarm condition. The authoring process must embed the watermark signals with sufficient frequency so as to satisfy the most demanding playback devices.

Even if there is no change in the embedded watermarks detected by the playback device, after a certain amount of viewing as determined by the appropriate metric of say, time or footage, the playback device may require an authorization from the CA smartcard if play is to continue. The CA smartcard does not need to process or even receive the entire content stream, but rather only the embedded information from the watermarked snippets the display device detects. So instead of decrypting content for the TV monitor, the CA smartcard becomes an authorized licensing device. The legitimate playback device inspects content to see if a playback license is required, and the playback device refuses to play marked content if no license is presented.

The watermarking of the content done during authoring may be done without regard to the billing infrastructure as long as the embedded signals differ across content so as to allow proper granularity when reconciling the billing process. If every title is marked distinctly, this will allow for later arbitrary assignments of billing units.

Freshness considerations must be addressed so that presentation of licenses or authorizations by the CA smartcard cannot be effectively replayed so as to circumvent appropriate billing for multiple viewings of the same content. For example, the playback device may include a random number along with the watermark in the authorization request..

Compromise of this system requires the pirate to corrupt the watermark while still maintaining product quality, or produce playback devices that ignore the watermark, or compromise individual customers' CA

smartcards. This should be considered an evolutionary approach to security, in that as watermark technology improves, new chargeable content can be released with the newest watermarks embedded as well as with the more vulnerable previous types of watermarks [9]. This allows for backwards compatibility with a population of display devices which shrinks as a percentage of the customer base of compliant devices as new more fully-featured display devices continually enter the marketplace.

The binding authority is essentially a certification authority and many techniques are known for preventing a single source of compromise for signing operations. For example, a hierarchy of signing authorities may exist so the root key is used infrequently [Error! Reference source not found.]. Furthermore, any signing operation may really be the result of multiple signing operations, so the compromise of a single machine does not compromise the combined signature operation [Error! Reference source not found.]. Finally, via signature schemes using “proactive” security, the multiple signing operations must be temporally related, so the compromise of a sufficient number of the individual signing operations must be done within a single window, in order to compromise the signature operation [Error! Reference source not found.].

4. Related Work

Watermarks [2,9] carry an embedded signal within the desired signal and have been part of proposed solutions for many forms of copyright control, in particular copy protection. For example, a movie may carry a watermark that instructs a VCR that it is not a recordable signal. The focus of such work has been copy protection and not conditional access.

Watermarks differ from fingerprints [1]. Fingerprints are signals in content that enable tracing information about the particular use or instance of the content (e.g., the name of the purchaser). Fingerprinting must be resistant to collusion, since an attacker may compare and combine different instances of the content. In our application, however, a particular content will have a single signal representing its billing ID.

Linnartz [7], Linnartz, Depovere, Kalker [8], and Epstein [3] also discuss watermarking in the context of copy protection and conditional access.

[8] differentiates between playback control and recording control, where a pirate may disable watermark checking functionality within his own hacked recorder while finding it more difficult to produce content which plays back on standard (i.e., compliant) players. When dealing with the conversion from “one-copy” to “no-more-copy,” the authors do not want to provide a hook for hackers to break the system. Their proposal also addresses “never-copy” and “free-copy” (i.e., no watermark) material.

They utilize electronic recognition of the storage medium (ROM vs. RAM), a “physical mark” P embedded on the disc which cannot be read or recovered externally of the drive, and authorization tickets T . One goal is to prohibit playback of “never-copy” content from non-original media. During mastering of the ROM disc, the manufacturer performs the operation $P=F(U)$ where U is a seed provided by (and only known to) the content owner and F is an appropriate one-way function. It is possible to have the construction of P from U designate specific publishers through appropriate design of the function F . It is intended that the physical mark reserved for ROM content is hard for a casual copier to insert on RAM discs.

It is claimed that a pirate publisher attempting to write a particular P in order to make a bit-exact copy of a copyrighted disc must tamper with the DVD press (which he may not own) if the press expects to be given the value of U which is not known to the pirate. According to Linnartz [7], one realization of such a physical mark P is the “wobble groove” in optical disks. If a disc contains a P reserved for professional content, and the content contains a watermark W , then playback requires that: $W=F(P)$ is satisfied in the case of never-copy; that the validation ticket T is present and that $T=F(P)$ and $W=F(F(T))$ are satisfied (where $F(T)$ replaces T at the output of the drive) in the case of one-copy; that $W=F(T)$ is satisfied in the case of no-more-copy. Playback is also allowed if the disc contains a physical mark reserved for recordable media and the content contains a valid W which is used for professional recording and the validated one-copy T is present and $W=F(T)$. During recording, the compliant recorder passes the ticket through the one-way function F before transferring it to disc. Recording of copyrighted content is allowed only if $W=F(F(T))$.

5. Conclusion

This paper presented a non-cryptographic CA system that protects content both against conditional access (initial purchase) threats, and copy protection (redistribution or repeated access) threats. The CA smartcard functions as a license authority that is able to authorize display devices to display protected content. Since the display device will only accept authorizations from the paired CA smartcard, the pirate cannot build a counterfeit licensing authority based on his smartcard and sell it to his customers [4,6]. These CA smartcards need not contain universal (system-wide) secrets, making them less attractive security targets.

The efficacy of this system depends upon both the robustness of watermarking and the prevalence and robustness of display devices that detect watermarks and require authorization. How can this detection and authorization capability become standard in display devices? One technique is for it to be bundled with licensed technology that is a desirable feature or to make it part of a being a compliant product (e.g., in DVD systems, CSS security is part of the DVD specification).

Acknowledgments

This paper benefited greatly from discussions with Michael Epstein and from the comments of the anonymous referees.

References

1. D. Boneh, and J. Shaw. "Collusion-secure Fingerprinting for Digital Data," *Advances in Cryptology: Proceedings, CRYPTO '95*, Springer-Verlag, 1995.
2. I.J. Cox, J. Kilian, T. Leighton, and T. Shamoon. "A Secure Robust Watermark for Multimedia," *Workshop on Information Hiding*, University of Cambridge, Springer-Verlag, 1996.
3. M. Epstein. "The Use of Watermarking to Protect High Value Watermarking Material," *ATSC*, 7/22/98.
4. 'Open Verifier' Functionality in Consumer Electronics Devices, GD-T204, Release B, News Data Systems, Ltd.
5. ISO 7816 Identification Cards, Integrated Circuit Cards with Contact, 1987.
6. D.M. Goldschlag and D.W. Kravitz. "Pirate Card Rejection," *Cardis 98*, Louvain-la-Nueve, Belgium, September 14-16, 1998.
7. J.P.M.G. Linnartz. "The 'Ticket' Concept for Copy Control Based on Embedded Signalling," *ESORICS 98*, Louvain-la-Nueve, Belgium, September 16-18, 1998.
8. J.P. Linnartz, G. Depovere, and T. Kalker. "Philips Electronics Response to Call for Proposals Issued by the Data Hiding Subgroup Copy Protection Technical Working Group," July 1997.
9. F.A.P. Petitcolas, R. J. Anderson, and M.G. Kuhn. "Attacks on Copyright Marking Systems," *Second Workshop on Information Hiding*, Portland, Oregon, LNCS 1525, pages 218-238, Springer-Verlag, April, 1998.
10. CCITT, Recommendation X.509, "The Directory-authentication Framework," Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.
11. C. Boyd, "Digital Multisignatures," *IMA Conference on Cryptography and Coding*, Clarendon Press, pages 241-246, (Eds. H. Baker and F. Piper), 1986.
12. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, M. Yung, "Proactive Public-Key and Signature Schemes," *The 4th ACM Symposium on Computer and Communications Security*, April, 1997.

Providing authentication to messages signed with a smart card in hostile environments

Tage Stabell-Kulø, Ronny Arild, and Per Harald Myrvang

Department of Computer Science

University of Tromsø

Tromsø, Norway

{tage,ronnya,perm}@pasta.cs.uit.no

24. March 1999

Abstract

This paper presents a solution to how a smartcard can be used to sign data in a hostile environment. In particular, how to use a smart card to make a signature on data when the machine to which the smart-card reader is attached can not be trusted. The problem is solved by means of a verification server together with a substitution table and a one-time pad; it is argued that lacking a trusted channel from the card, our solution is minimal.

An invalid signature (a signature on data not intended to be signed) can only be made if the online server colludes with the machine the user is using. In all other circumstances, only a denial-of-service attack is possible. The realization is applicable in practice, but slightly awkward.

1 Introduction

It is difficult to digitally sign data in a hostile environment, even armed with a smart card that can create digital signatures by means of some public-key technology [3]. Assume a user P sitting in front of a machine M with the smart card residing in a smart-card reader connected to M . If P wants to sign X , he has no means to verify that M actually gives X to his card; thus, P can not use the card without trusting M just as much as he trusts the integrity of the card, in which case he could use M to sign rather than involving a smart card in the first place.

The problem is that there is no authenticated “channel” from the card to the user. The card is unable to “tell” P what it is about to sign, and P can not verify that X has been received for signing [1]. The problem is

well known [4, 14]. Authenticated channels can be obtained by means of, for example, more powerful hardware, such as contemporary PDAs. With this type of hardware, integrity is obtained since the PDAs have a (small) display on which X can be shown. However, smart cards are prevalent and we seek a solution using this technology.

In general, data integrity relies on either secret information or authentic channels [7]. In other words, when using smart cards without any authentic channels, some sort of secret information is needed.

There are many settings where one might desire to sign data with a smart card, where the environment might be hostile. For example a point-of-sale terminal, or during a visit to an “Internet café”. Using a computer laboratory at a university is another example. In general, any environment where one does not want to include M in the trusted computing base (TCB), for whatever reason [13].

The paper is outlined as follows. Section 2 is concerned with describing the system and our solution. Then, in Section 3, our protocols are shown and analyzed. Section 4 discusses related work. In Section 5 we present conclusions and give directions for future work.

2 Overview

This paper examines a particular—albeit common—setting where smart cards are employed. The general idea is that the user P has some data, an email perhaps, that he wants to sign, using the secret key stored in his card. He would instruct the software running on M to send the data to the smart-card reader, insert his card, and having the signature returned in order to be attached to the email. The problem is that M might give any data

to the card, and the card will sign. Unless P can verify public-key signature in his head, he has no means to judge whether M is trustworthy or not.

In fact, what seems to be a single problem really poses three distinct challenges:

1. How can P ensure that the correct data has been signed?
2. How can P verify that the signature is valid?
3. Is it possible for a third party to conclude that P has verified that the correct data has been signed?

The last is required if the signature P makes with his card is to have a non-trivial value.

Concerning the first question, only P knows the answer to this, since only he knows what he intended to have signed; the fact that M also happens to know is of no relevance to us because M is not trusted. The user must thus be involved in providing an answer to the first question. Or, in other words, no solution to this problem can be envisioned without involving the user in some way, after the signature has been made.

In a realistic scenario, we can rule out the possibility of P verifying the signature himself. This implies that a third party must verify the signature itself. Such a third party should take the form of an online service, in order to better enable the user to timely obtain an answer to the second question. This, however, raises a new obstacle: How can this online service, called O , communicate with P over a channel that provides integrity? Our contribution is a working method to solve this particular problem.

Turning now to the third challenge, it will become evident that P can sign a certificate that, together with the credentials our solution creates as it progresses, enables others to conclude that the data indeed was signed by P 's card, with P 's consent. It might be worth noting that we are only interested in signing. P is unable to encrypt anything on his smart card without trusting M . That is, secrecy can not be obtained at all in the setting we describe.

Our solution consists of three parts, an on-line service, a small one-time pad (an OTP) and a shared secret. In the following we will describe how some data is signed by means of a smart card in a hostile environment (details are given in Section 3). We use the notation from the BAN logic [2],

1. The machine M is not trusted. Thus, M is not the logical sender or recipient of any message (even

though the actual hardware will be used to send messages). From a logical point of view, M is part of the communication infrastructure. In this light, the only principals of interest are the user P , his smart card C and the on-line service O (to be described below).

2. The user has some data X , which typically is a string of characters (i.e., a text). P inserts his card (into the smart-card reader attached to M) and instructs M to transfer the data to the card. The card is then instructed to sign the data it received.

1: $P \rightarrow C : X$

3. The card accepts the message and signs X , creating $\{X\}_{K_C^{-1}}$. Notice that C has no means to verify that X actually originates from P . As mentioned above, two questions must be answered:

- (a) Is the signature valid?
- (b) Has the correct data been signed?

The online service can be used to verify the signature's validity; the signed data is sent to O .

2: $C \rightarrow O : \{X\}_{K_C^{-1}}$

4. The crux of our solution is that O can send back to P a transformation f of the data it has verified. Assume that P and O share a *small* secret one-time pad and a secret number. After verifying the signature on X , O will create two new messages as follows.

Using the one-time pad, a new message $Z = f(X)$ is constructed, and sent to P .

3: $O \rightarrow P : Z$

Z is thus the message X transformed for integrity under a one-time pad. We will discuss this transformation below.

If the signature is valid, O constructs a certificate asserting this fact. The certificate is sent to a public server of some sort. We call this server S , its existence is only for convenience and might very well be O itself.

A random number Y is associated with each one-time pad. Y is known only to P , but $H(Y)$ is known also by O . If O finds that the signature is valid, O will sign a certificate stating this fact; the certificate will include $H(Y)$. By releasing Y , P proves that he accepts the signature.

4: $O \rightarrow S : \{C, X, H(X, H(Y))\}_{K_O^{-1}}$

5. When Z is received by P , he can without much effort (and without using M to anything but display Z) verify that $Z = f(X)$. Since Z is a transformation of X , P can conclude that the content was what he intended to sign, and that his trusted server O has verified the signature. P now releases Y by sending it to S .

5: $P \rightarrow S : Y$

To sum up, O verifies the signature made by C , and P acknowledges the actual text by releasing Y .

Up to this point we have used logical messages. If we look at the actual implementation we find eight messages being sent over various channels; see Figure 1. The messages can be described as follows:

Message 1: $P \rightarrow M : X$
 Message 2: $M \rightarrow C : X \text{ from } P$
 Message 3: $C \rightarrow M : \{X\}_{K_C^{-1}}$
 Message 4: $M \rightarrow O : \{X\}_{K_C^{-1}} \text{ from } C$
 Message 5: $O \rightarrow S : \{C, X, H(X, H(Y))\}_{K_O^{-1}}$
 Message 6: $O \rightarrow M : \langle X \rangle_{OTP}$
 Message 7: $M \rightarrow P : \langle X \rangle_{OTP} \text{ from } O$
 Message 8: $P \rightarrow S : Y$

Messages 6 and 7 contains the string of digits O has constructed based on its copy of the OTP. Since the OTP is secret, the string is X combined with a secret. In BAN such a construction is denoted as $\langle X \rangle_{OTP}$.

Section 3 gives a detailed description of the small one-time pad that is required, a closer look at the messages that are sent and, most important, a careful analysis of the logical meaning of each message and of the certificates that are required to conclude that X was signed by C with the consent of P .

3 Signing

This section starts out by presenting the details of the one-time pad. Being small it is suited for practical use; Section 3.1 discusses it. Section 3.2 is concerned with a theoretical analysis of the certificates and credentials required to assert that a signed statement from a smart card logically originates from the user that control the card. In Section 3.3 we discuss the trusted computing base of our solution. Section 3.4 describes the implementation status, and gives a preliminary performance analysis.

3.1 The one-time pad

We assume that P does not have any significant computational resources at hand (M can not be trusted). Since it is unreasonable to assume that any user can verify digital signatures without the help of a computer, we must thus construct a secure channel from O to P , on which a message can be sent. That is, P needs to receive from O some information that convinces him that the correct text was signed. This information must be a function of the message X in order for P to know that the correct text has been signed. In addition, P must be convinced that the message he receives comes from O . Taken together, the channel we are about to construct must provide authentication (since authentication implies integrity [7]). Clear-text attacks are indeed a threat since M knows X .

If, on the other hand, X was unknown to M then P and O could share a list L of random numbers, each number L_i of L being as long as X . O would verify the signature on X , calculate $Z = X + L_i$ and send the result to P . P would be able to calculate $Z - L_i$ and verify that the card had signed X . This cipher would be perfectly secure [9].

In our system, each OTP contains two small tables. The first contains random numbers, as one would use to create a one-time pad. However, in our case X is known, and this procedure alone offers no integrity at all. This is so because if 'A' and a random number yields 12 then 'B' must have yielded 13. We overcome this problem by incorporating an additional table. It is a permutation of the characters; we denote this a *substitution table*.

We now describe the OTP used by P and O . In the current implementation, the alphabet available to P are all the upper-case characters, space (denoted as ' '), dot ('.'), the digits and the two symbols \$ and @; 40 characters in all. These characters are matched with a table of random numbers, assigning a random number to each character. Appendix 1 shows two examples of tables; each has six rows:

Letter: The alphabet available to users

Subst: The substitution table; each character from the alphabet is replaced by the corresponding number from the substitution table.

X: In this row the user writes his message

OTP: The number representing each character is added (modulo 40) to the corresponding element in the One Time Pad.

Z: The result.

Y: A secret number, see below.

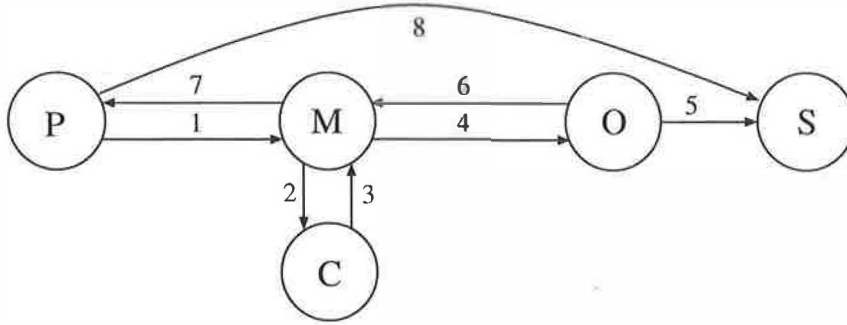


Figure 1: The protocol run

When P receives Z from O , he would want to verify the result. In order to do so, he proceeds as follows.

1. Count the number of characters in the message, and prepend this number (as a string) to the message.
2. Write the string in the table (in the row marked X) above the random numbers.
3. For each character, add the ordinality of the character (taken from the substitution table) with the random number. The addition must be done modulo 40 (the number of characters).

An example of a table which is filled in is shown in the Appendix. The string "GIVE TAGE@ACM.ORG \$500." is encrypted for authentication.

If P sees that Z indeed is the correct transformation of X , he will release Y . The certificate generated by O contains $H(Y)$, but Y is only known to P . In other words, by releasing Y , P makes it known that he supports the certificate issued by O .

3.2 Theory

The security of our system hinges on three properties. The first is that one component in each sum is a random number. Randomness ensures the resulting list of numbers are random. No amount of calculation or number of previous messages can give information necessary to alter the text. Second, each OTP and substitution table can only be used once. Third, text can not be appended to the string.

Obviously, the length of the string that can be transmitted (and verified) in this manner is restricted by the length of the pad. However, the pad can be made as long as one desires and the amount of work to verify a message increases linearly with length. Another way to

Message	Meaning
X	X
$\{X\}_{K_C^{-1}}$	C says X
$\{C, X, H(X, H(Y))\}_{K_O^{-1}}$	$O C$ says X , $O Y$ says X
Y	Y

Table 1: Messages and their interpretation

increase the task of verification is to increase the alphabet length (now being 40). If this length is increased, the OTPs and corresponding substitution tables must be increased accordingly.

We have described how a user P can sign a message; we now describe how a receiver verifies that a signed message is valid. Assume a user Q receives a message $\langle Y, \{X\}_{K_C^{-1}} \rangle$ from P . Assume furthermore that Q believes that C belongs to P . Upon receiving the message, Q contacts S and asks for the certificate that O should have generated. Obtaining it, Q has all he needs to conclude that X was signed by C , that O has verified that the signature was in order, and that P has verified that the correct data was signed. The four datums that are available to Q is shown in the left column of Table 1. Informally, the fact that P has released Y is proof that P has verified the signature. We will now give a more formal view of the system, using the theory from [6].

If Q is to act upon X he would need a certificate, signed by P (or a principal Q believes speaks for P), asserting that possessing the four items together vouches for the conclusion that X originates from P . Since M is not trusted, P does not control C , and the assumption $C \Rightarrow P$ is unwarranted. The intention of P is that no-one will hold him responsible for any message X unless the following conditions are met:

- X is signed by C .

- The signature made by C is verified by O . O must say that C have said X .
- O must tie (the secret) Y to the signed message. This enables P to accept the signature by releasing Y .
- Y is available.

All this is captured in the following certificate

$$P \text{ says } (C \wedge O|C \wedge O|Y) \Rightarrow P \quad (1)$$

Since Y is secret, Q is unable to satisfy the certificate (1) unless P releases Y . In practice, S could in addition act as an on-line verification for the validity of C in that P would make C issue $C \text{ says } (S|C \wedge C) \Rightarrow C$, see [6] for details.

With these credentials, the axioms and inference rules set forth in [6], it follows that $P \text{ says } X$. Note that the use of Y give the message the properties of a *transaction authentication* as defined in [7]; message authentication and the use of time-variant parameters (timeliness or uniqueness).

3.3 Trusted Computing Base

As can be seen from (1) it is a prerequisite for certificate verification that O says that Y says X . However, P does not want to include O in his TCB. O has not been given Y by P , but rather $H(Y)$. When O quotes Y as saying X it might turn out that O is mistaken; this is in fact correct, in the cases where M , for example, mounts some attack. In other words, when Q collects credentials he might or might not be able to locate Y . In such a situation there are two possibilities: Either P has not released it (he has detected an attack) or Y has been delayed or deleted as part of a traditional denial-of-service attack. Since P is at the mercy of M , there are no means to defend P against denial of service.

O is a *trusted third party* in that P trusts O to act according to the protocol (not to certify that a signature is good if it is not). On the other hand, O is not able to deceive P without colluding with M . When O and M colludes, M can feed a false message to the card and let O send an erroneous message back to P . The important issue is that alone, O can not deceive P . In the same manner as O is not in the TCB, neither is S , nor C . No principal is in a situation to make P release Y , which will erroneously make (1) is true, without colluding with some other principal.

3.4 Performance

The system described is not yet fully implemented, although the infrastructure is; this includes a verification server, certificate and signature handling, and a cryptographic strong random number generator. We have, however, done preliminary performance tests using pre-defined OTPs and substitution tables.

Experiments show that verification is initially done at a speed of approximately 7–8 seconds per character. However, speed increases as one gets accustomed to the calculation. Experienced users spend approximately 3–4 seconds per character. Slightly slower than typing, but in our opinion worthwhile.

4 Related work

Our solution is basically a Message Authentication Code (MAC). MACs are well covered in the literature, see for example [10, 7, 11]. However, most MACs are computationally intensive. Most types of MACs, such as MD5 [8], are surjective, and require some computation to be secure (mapping one language onto a smaller while being a one-way function).

The use of unconditionally secure MACs are described in [11, Chapter 10] with the use of orthogonal arrays (OAs). These OAs seems, however, to be infeasible to work with for human beings compared to substitution tables and OTPs that only require the use of elementary arithmetics.

Authentication by means of a secret one-time pad is an old invention [5]. In this article, we have combined the one-time pad with the release of a secret to authenticate that the verification of the signature.

5 Conclusion

We have shown that users can achieve secure authentication to messages signed with a smart card in hostile environments, using a partial trusted verification server together with a substitution table and a one-time pad. The applicability lies in that short messages with small character sets.

5.1 Future work

We are working on implementing the system described in this paper, using Cyberflex Open16K smart cards¹ that run a Java VM, based on the Java Card 2.0 specification [12]. Since these cards come without a coprocessor (for efficiently doing computations on large integers), signatures made with public key schemes such as RSA or ElGamal would be hard to implement efficiently. We are looking into this issue. In addition, we are also working on eliminating O from the protocol by storing OTPs and substitution tables on the smart card itself (this would require about 100-150 bytes of storage capacity for each OTP/substitution set). Making the verification process easier for end-users is also prioritized; using arrow keys to decrypt the message from O may be a workable solution. This is not a new idea; the use of arrow keys for decrypting OTPs was suggested in [14] and was based on the insertion of '+' and 'nextdigit' operations described in [1].

A way to eliminate the awkwardness of using OTPs and substitution tables would be to insert a channel between O and P for Message 6 in the protocol run. A mobile telephone could be used for this purpose, where O sends X and Y to P through, for example, the GSM network. P would then see that the message received on the phone's display corresponds with the message that was originally written by P . A drawback here is that P and O must share the secret Y , since M could otherwise send X to P (through GSM). This implies that P must trust that O does not release Y until P does it. This drawback, combined with that O can no longer be eliminated from the protocol, would probably not detract from the fact that P no longer needs to do substitutions and arithmetics in his head (or use arrow keys).

Acknowledgements

We would like to express our gratitude towards the other members of the PASTA project. Funding has been received from the Royal Norwegian Research Council through the GDD project (project no. 112577/431).

References

- [1] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-

¹Information about these smart cards is available at URL:<http://www.cyberflex.slb.com/>.

cards. *Science of Computer Programming*, 21(2):93-113, October 1993.

- [2] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18-36, February 1990.
- [3] Henry Dreifus and Thomas Monk. *Smart Cards - A Guide to Building and Managing Smart Card Applications*. IEEE Computer Press, 1997. ISBN 0-471-15748-1.
- [4] H. Gobioff, S. Smith, J. D. Tygar, and B. Yee. Smart Cards in Hostile Environments. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, Oakland, CA, November 1996.
- [5] David. Kahn. *The Codebreakers: The story of secret writing*. Macmillan Publishing Company, New York, USA, 1967.
- [6] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, 10(4):265-310, November 1992.
- [7] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., 1997. ISBN 0-8493-8523-7.
- [8] R. L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992.
- [9] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656-715, October 1949.
- [10] G. J. Simmons, editor. *Contemporary Cryptology: The Science of Information Integrity*. IEEE Press, 1992. ISBN 0-87942-277-7.
- [11] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, Inc., 1995. ISBN 0-8493-8521-0.
- [12] Sun Microsystems, Inc. Java Card 2.0 Language Subset and Virtual Machine Specification. Revision 1.0 Final, October 1997.
- [13] US Department of Defence. *Trusted Computer System Evaluation Criteria*, 1985. DOD 5200.28-STD.
- [14] B. Yee and D. Tygar. Secure Coprocessors in Electronic Commerce Applications. In *Proceedings of The First USENIX Workshop on Electronic Commerce*, New York, New York, July 1995.

Example OTP and substitution table

Letter	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z _ . \$ %
Subst	05 27 13 32 03 21 16 22 00 08 26 06 04 07 18 39 30 15 19 09 17 33 24 38 17 25 14 20 10 02 31 33 34 35 12 01 36 28 11 29
X	2 3 G I V E _ T A G E @ A C M . O R G _ \$ 5 0 0 .
OTP	31 25 08 32 02 16 38 18 19 13 17 01 37 38 20 24 00 33 10 01 24 34 37 11 01 05 08 14 15 29 03 03 18 39 30 05 10 22 24 14
Z	04 17 38 11 35 34 34 20 05 03 35 30 23 02 04 12 17 13 00 37 35 15 02 16 29
Y	0x8bde94b630f1504b

Letter	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z _ . \$ %
Subst	05 27 13 32 03 21 16 22 00 08 26 06 04 07 18 39 30 15 19 09 37 23 24 38 17 25 14 20 10 02 31 33 34 35 12 01 36 28 11 29
X	
OTP	31 25 08 32 02 16 38 18 19 13 17 01 37 38 20 24 00 33 10 01 24 34 37 11 01 05 08 14 15 29 03 03 18 39 30 05 10 22 24 14
Z	
Y	0x8bde94b630f1504b

Authenticating Secure Tokens Using Slow Memory Access

John Kelsey Bruce Schneier

{schneier,kelsey,schneier}@counterpane.com

Counterpane Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419

Abstract

We present an authentication protocol that allows a token, such as a smart card, to authenticate itself to a back-end trusted computer system through an untrusted reader. This protocol relies on the fact that the token will only respond to queries slowly, and that the token owner will not sit patiently while the reader seems not to be working. This protocol can be used alone, with “dumb” memory tokens or with processor-based tokens.

1 Introduction

Smart cards have been used in many applications that require that the data be secured from the cardholder. In the Modex stored-value smart card system, for example, the smart card stores a particular monetary value. An attacker who can successfully change the value stored in the card can essentially print money.¹ In GSM phones, smart cards provide identity information used to prevent cloning and in billing. An attacker who can modify that information can charge cellular phone calls to another account [?]. Smart cards used to protect satellite television can be attacked to obtain free services [McC96].

These cards are vulnerable to a large class of attacks [And94, Sch97, Row97, Sch98], from reverse-engineering [AK96] and protocol failures [AN95] to side-channel attacks [Koc96, Koc98, KSWH98, DKL+99] and fault analysis [BDL97, BS97]. In all of these attacks, the attacker can defeat the security of the card by breaching its secure perimeter and learning the confidential information stored within. With reverse-engineering, the attacker defeats the tamper-resistance measures directly; with side-channel and fault-analysis attacks, the attacker exploits weaknesses in the physical security to learn information within the secure perimeter.

¹For other examples, see [CP93], [SK97], and [RKM99].

There is another class of applications for smart cards—applications in which the value of a successful attack is much smaller. These cards might provide micropayment information, low-security access control, or act as payment vehicles in circumstances with low marginal cost of goods (pay television, public transportation, etc). In these applications we are less concerned with individual fraud, and more concerned with an organized attack to create and distribute fake cards. Someone who sneaks onto the subway or watches a satellite movie without paying isn't going to affect the service provider's bottom line, but someone who is able to counterfeit access tokens that allow everyone to sneak onto the subway or watch the movie could collapse the entire system. Hence, the primary threat is not from an isolated attack against a single card, but an attack that can be scaled to multiple cards.

The threat is from untrusted card readers that the user may stick his card into. In this paper, we propose a low-tech authentication protocol that is useful in this sort of situation. Our protocol makes use of what we will call a “slow memory device”: a device that responds to queries by revealing the contents of different memory locations, but one that necessarily takes several seconds to do so.

The protocol relies on the fact that the cardholder does not have infinite patience. If he puts his smart card into a reader and nothing happens for several seconds, he will likely pull the card out and try again. If nothing happens again, he will find another reader. The slow response of the card ensures that a fraudulent reader can only do so much damage before the cardholder removes his card.

This protocol makes no cryptographic assumptions, and is independent of any cryptography that may or may not be in the system. It can be implemented by itself, or in conjunction with cryptographic controls.

The rest of the paper is organized as follows. In Section 2 we describe the slow memory device and its functionality. In Section 3, we describe our authentication protocol. In Section 4 we discuss variants

and extensions to this basic protocol, and in Section 5 we discuss applications.

2 The Slow Memory Device

This protocol assumes the existence of a slow memory token. By this we mean a token—a smart card, a Dallas Semiconductor iButton, etc.—that acts as a memory device, but never responds to a request in less than t seconds ($t = 10$, for the purposes of this example). It is impossible for a terminal to get more information during that time; the token's electronics are such that it simply cannot respond to requests faster.

This memory token has m memory locations, each w bits wide ($w = 64$ for the purposes of this example). The token does not need a processor, nor does it need to implement any cryptographic primitives in order to execute this protocol.

3 The Basic Protocol

There are three parties in this protocol:

- **The Token:** The slow memory device.
- **The Terminal:** The untrusted card reader.
- **The Trusted Machine:** A trusted computer connected to, and possibly remote from, the Terminal.

Our authentication protocol is simple. A user inserts his Token into a Terminal. The Terminal now needs to prove to a Trusted Machine that the Token is currently inserted into the Terminal.

The Terminal is only marginally trusted, and could be malicious. In order to complete the protocol correctly the Terminal must be connected, via a secure data link, to a Trusted Device. The Trusted Device keeps an exact copy of the contents of the Token; this is a shared secret that the Terminal does not know.

Our protocol is as follows:

- (1) The holder of the Token inserts it into the Terminal.
- (2) Terminal reads the header information from the Token.

- (3) Terminal sends this information over an encrypted link to the Trusted Device.
- (4) Trusted Device generates a random challenge and sends it back over the encrypted link to the Terminal. This challenge consists of a list of n memory locations on the Token.
- (5) Terminal reads the n memory locations from the Token, XORs them all together, and sends the result back over the encrypted link to the Trusted Device.
- (6) The Trusted Device verifies this information; if it checks out, it believes that the Token is currently inserted into the Terminal.

Note that there are no cryptographic primitives in the protocol: the only mathematical operation involved is XOR. There are no encryption functions, one-way hash functions, or message authentication codes used in the protocol.

The Token might have 1000 64-bit memory locations. Each time read request comes in, it takes ten seconds to respond, and without reverse-engineering the device and tampering with its internals, this can't be made any faster. Assuming ten seconds per communications exchange, and setting $n = 1$, steps (2) and (3) take ten seconds, step (4) takes another ten seconds, and step (5) takes another ten seconds. This gives us a total of 30 seconds per transaction. Now, this can be extended a little bit without the user knowing. A malicious Terminal might ignore the protocol completely, and simply query the Token (repeat step (5)). Maybe the Terminal can perform six queries instead of one, doubling the transaction time, before the Token owner removes his card.

To provide adequate security, we need to account for possible malicious behavior by the Terminal in how many transactions are allowed, keeping the probability of success acceptably low even if most Terminals are corrupt.

3.1 Reasonable Values for n

Suppose the Token has m memory locations, that each instance of the protocol requests the XOR of n random locations, and that $n \ll m$. Assuming an attacker eavesdrops on each authentication, he will learn the contents of n memory locations, not all of them different, after each transaction. The average number of authentications that the Token can accomplish before the attacker has a better than

0.5 chance of impersonating the Token (that is, being able to respond correctly to a random query), is:

$$\log(n)/(\log(m) - \log(m - n))$$

For example, for $m = 1000$ and $n = 5$, an attacker who eavesdrops on 322 authentications has a better than 0.5 probability of being able to impersonate the Token by answering a random challenge correctly.

This, of course, is not the most effective attack. A fraudulent Terminal will specifically query the Token to learn the memory locations that it does not know. Hence, a malicious Terminal can learn the entire contents of a Token in m/n queries. So for the parameters above, a malicious terminal that conducts 100 fraudulent transactions will have a better than 0.5 probability of being able to impersonate the Token. The Token owner, though, would have to be convinced to allow the Token to be used in 100 ineffectual transactions.

4 Extensions

4.1 A Button on the Token

Ideally, we'd have some user-interface mechanism on the Token, like a pushbutton. The Token is willing to perform only once per button-push. Alternatively, the Token buzzes or lights up once per memory query answered. This would make multiple queries much harder for the Terminal to make, since the Token owner could detect that the protocol was not proceeding as specified.

4.2 Reducing Storage Requirements in the Trusted Device

As written, the Trusted Device must store a complete copy of the Token's memory location. If this is too much memory, the Trusted Device could store the Token's ID information and a secret key known only to it (and not the token). The Token's memory locations would then be the memory address encrypted with this secret key, and would have to be loaded onto the Token by the Trusted Device.

4.3 CRC Hardware on the Token

If the Token can afford CRC hardware, then queries can be handled using this alternate protocol:

- (1) The holder of the Token inserts it into the Terminal.
- (2) Terminal reads the header information from the Token.
- (3) Terminal sends this information over an encrypted link to the Trusted Device.
- (4) Trusted Device generates a random challenge and sends it back over the encrypted link to the Terminal. This challenge consists of a list of n memory locations on the Token.
- (5a) The Terminal sends the Token a request for the n memory locations.
- (5b) The Token goes through the motions (and delay) of sending it out internally, but only outputs the 32-bit CRC of the n requested memory locations.
- (6) The Trusted Device verifies this information; if it checks out, it believes that the Token is currently inserted into the Terminal.

Each memory location can now be 32 bits long, and even one unknown memory location in the query string prevents an attacker from succeeding in an impersonation attack. Note that this system works best if there are lots of Terminals under different entities' control. If a Token only interacts with one Terminal every time it executes the protocol, then this system doesn't work very well.

4.4 Incrementing Values in the Token and Trusted Device

If we're worried about an attacker reverse-engineering and making lots of copies of the Token, then we have each query of memory location cause that memory location to increment by one, modulo 2^{32} , or rotate left one bit, or whatever else is cheap enough to be implemented. Note that this update must occur on both the Token and the Trusted Device, and assumes that there is either only one Trusted Device or that the different Trusted Devices can communicate with each other securely to synchronize these updates.

This variant does not help against impersonation attacks. What it does do is to make continued synchronization of those counterfeit Tokens very expensive and complex. Now, if any memory location in the challenge string has been changed, the duplicated Token fails to give the correct answer.

4.5 Reducing the Amount of Trust in the Trusted Device

A given Trusted Device may be only partially trusted. Instead of having it store a complete listing of the Token's memory locations, it can be given a series of precomputed challenges in order and the right responses to be expected. (If we use the previous extension, this will work only for small numbers of different Trusted Devices.)

5 Security Analysis

The slow memory protocol is designed to frustrate a particular kind of attack. It is intended to provide security against an insecure reader trying to collect enough information from a token to be able to impersonate it. It does not provide security against someone reverse-engineering the token and cloning it (although the extension described in Section 4.2 considerably frustrates that attack in most circumstances), nor does it provide security in the event that the back-end database (the Trusted Device in the protocol) is compromised.

A more extensive security analysis will be in the final paper.

6 Applications

A complete discussion of applications, along with their security ramifications, will be in the final paper.

The most obvious applications are things like non-duplicable keys for locks and alarms, free passes or one-day (or k -day) coins, and login keys. In these applications, we are less concerned with a single person hacking the authentication device than the same single person being able to distribute a large number of them.

One of the most beneficial aspects of this protocol is that it works well with cryptography, even though it has no cryptography itself. We can use a message authentication code such as HMAC [BCK96] in addition to this protocol, and if the MAC breaks, the memory trick still works. Or course, all Trusted Devices must be trusted with copies of the memory maps.

7 Conclusions

Security countermeasures must be commensurate with the actual threats. In this paper we have presented a low-tech security solution that helps mitigate a specific threat, and can be used in conjunction with other cryptographic countermeasures.

8 Acknowledgments

The authors would like to thank Chris Hall for his helpful comments. Additionally, the authors would like Niels Ferguson, who broke the MAC and subsequently inspired this research.

References

- [And94] R. Anderson, "Why Cryptosystems Fail," *Communications of the ACM*, v. 37, n. 11, Nov 1994, pp. 32–40.
- [AK96] R. Anderson and M. Kuhn, "Tamper Resistance – A Cautionary Note," *Second USENIX Workshop on Electronic Commerce Proceedings*, USENIX Press, 1996, pp. 1–11.
- [AN95] R. Anderson and R. Needham, "Programming Satan's Computer," *Computer Science Today: Recent Trends and Developments*, LNCS #1000, Springer-Verlag, 1995, pp. 426–440.
- [BCK96] M. Bellare, R. Canetti, and H. Karwczuk, "Keying Hash Functions for Message Authentication," *Advances in Cryptology — CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 1–15.
- [BDL97] D. Boneh, R.A. Demillo, R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Advances in Cryptology—EUROCRYPT '97 Proceedings*, Springer-Verlag, 1997, pp. 37–51.
- [?, BGW98] M. Briceno, I. Goldberg, D. Wagner, "Attacks on GSM security," work in progress.
- [BS97] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Advances in Cryptology—*

- CRYPTO '97 Proceedings*, Springer-Verlag, 1997, pp. 513–525.
- [CP93] D. Chaum and T. Pederson, “Wallet Databases with Observers,” *Advances in Cryptology — CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 391–407.
- [DKL+99] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestre, J.-J. Quisquater, and J.-L. Willerns, “A Practical Implementation of the Timing Attack,” *CARDIS '98 Proceedings*, Springer-Verlag, 1999, to appear.
- [Koc96] P. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems,” *Advances in Cryptology—CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 104–113.
- [Koc98] P. Kocher, “Differential Power Analysis,” available online from <http://www.cryptography.com/dpa/>.
- [KSWH98] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, “Side Channel Cryptanalysis of Product Ciphers,” *ESORICS '98 Proceedings*, Springer-Verlag, 1998, pp. 97–110.
- [McC96] J. McCormac, *European Scrambling Systems*, Waterford University Press, 1996.
- [RKM99] C. Radu, F. Klopfert, and J. De Meester, “Interoperable and Untraceable Debit-Tokens for Electronic Fee Collection,” *CARDIS '98 Proceedings*, Springer-Verlag, 1999, to appear.
- [Row97] T. Rowley, “How to Break a Smart Card,” *The 1997 RSA Data Security Conference Proceedings*, RSA Data Security, Inc., 1997.
- [Sch97] B. Schneier, “Why Cryptography is Harder than it Looks,” *Information Security Bulletin*, v. 2, n. 2, March 1997, pp. 31–36.
- [Sch98] B. Schneier, “Cryptographic Design Vulnerabilities,” *IEEE Computer*, v. 31, n. 9, September 1998, pp. 29–33.
- [SK97] B. Schneier and J. Kelsey, “Remote Auditing of Software Outputs Using a Trusted Coprocessor,” *Journal of Future Generation Computer Systems*, v.13, n.1, 1997, pp. 9–18.

SCFS: A UNIX Filesystem for Smartcards

Naomaru Itoi, Peter Honeyman, and Jim Rees
Center for Information Technology Integration
University of Michigan
Ann Arbor

itoi@eecs.umich.edu, honey@citi.umich.edu, rees@umich.edu

Abstract

Smartcard software developers suffer from the lack of a standard communication framework between a workstation and a smartcard. To address this problem, we extended the UNIX filesystem to provide access to smartcard storage, which enables us to use files in a smartcard as though normal UNIX files, but with the additional security properties inherent to smartcards.

1 Introduction

Today, it is easy to purchase smartcards in reasonable prices, *e.g.*, \$5 - \$20 for each. However, smartcard software development is hard: smartcard software developers have long suffered from the lack of a user friendly standard communication protocol between application software¹ and a smartcard. The ISO-7816 communication protocol [9] is so widely accepted that virtually all smartcards support it.² However, the protocol is not a particularly desirable one:

- It is a primitive message passing protocol. Providing only read and write opera-

¹ "Application software" is a program running on a workstation that communicates with a smartcard. A program running on a smartcard is called "on-chip software".

² Almost all smartcards support ISO-7816-1, -2, and -3; many also support ISO-7816-4 [18]

tions for raw data, it does not define high-level interfaces such as UNIX files and I/O streams. This hampers our ability to build application software.

- Although all smartcards support ISO-7816, details of implementation of the protocol differs among vendors and types of smartcards. This requires software developers to tailor their applications to specific smartcards.

Differences among smartcards range from trivial ones, such as different opcodes, to essential ones, such as different authentication mechanisms, *etc.* For example, the CLA byte of application class³ is 0x00 in some smartcards (Giesecke & Devrient STARCOS Version 2.1), while it is 0xc0 in others (Schlumberger MultiFlex).

To address the deficiencies of ISO-7816, many new standards have been proposed. Examples are:

- General purpose standards: Open Card Framework (OCF) [2, 8] and PC/SC [3, 4].
- Special purpose standards: PKCS #11 [12] for cryptography, EMV [5] and SET for electronic commerce [13].
- On-chip software standards: JavaCard [15] and MULTOS [16].

Although these standards provide abstractions at a higher level than ISO-7816-4, it remains a

³ See Guthery and Jurgensen [6] or ISO-7816 [9] for a description of "CLA" and "application class."

challenging task for developers to select a standard, purchase all software and hardware required, learn API and tools, and finally implement software. Furthermore, those standards do not eliminate problems with interoperability – *e.g.*, OCF limits the programming language to Java; PC/SC is used only with Windows – and create their own API dependencies, because software written for one standard does not run with another. We discuss these issues in Section 5.1.

Our solution to this problem is to embrace a classic, sophisticated API – the UNIX filesystem – instead of inventing a new one. The UNIX filesystem API suits a smartcard well because a smartcard is a passive device used for secure storage: a smartcard stores data (secrets), and responds to requests from a workstation to read or write the data. It does not initiate actions. This passivity is characteristic of storage devices such as hard disks. Cryptographic functions, such as get challenge, internal and external authenticate, verify key and PIN, are unique to smartcards. However, smartcards still act passively for these functions, and they are implemented with `ioctl()`.

In UNIX operating systems that support **vnodes** (equivalently, Virtual Filesystem, or VFS) [11] [14], it is possible to write a virtual filesystem that communicates with a special hardware device, *e.g.*, a smartcard, and mount it in the UNIX filesystem name space. The mounted hardware device then becomes identical to any UNIX filesystem hierarchy from the perspective of a user or application software. For example, if a smartcard is mounted on `/smartcard`, it is possible to use UNIX commands such as `ls`, `cd`, `pwd`, and `cat`, and system calls such as `open`, `read`, and `write` on files in the smartcard.

We have implemented a smartcard filesystem (or SCFS) in the OpenBSD-2.4⁴ kernel. With SCFS mounted, a user or an application can use files in a smartcard as she would normal UNIX files.

⁴OpenBSD is a free, 4.4BSD-based operating system. <http://www.openbsd.org>

The remainder of this paper is organized as follows. Section 2 describes our goals and the design of SCFS. Section 3 details implementation of SCFS. (Readers not interested in implementation details may want to skip Section 3.) Performance evaluation in Section 4 shows that the overhead of SCFS is small and does not substantially degrade the performance of smartcard software. We discuss SCFS with a comparison to other standards in Section 5. Future direction is described in Section 6 and concluding remarks are in Section 7.

2 Design

2.1 Design Goals

Our goal is to provide a user friendly interface to access a smartcard. We define design goals as follows, although not all can be achieved, for reasons outlined in Section 2.2:

- Files in a smartcard should be indistinguishable from other UNIX files.
- A smartcard can be accessed with any UNIX system call (*e.g.*, `creat`, `open`, `read`, and `write`).
- UNIX commands (*e.g.*, `ls`, `cd`, `pwd`, and `cat`) can be used to access files in a smartcard.
- The smartcard VFS must be able to access any smartcard that supports ISO-7816.
- The smartcard VFS should hide details about a smartcard to users.
- Security of a smartcard must be preserved.
- No smartcard files may be cached in the UNIX system because a smartcard is a more secure place to store data (see the end of Section 2.3).

2.2 Design Problems

A huge obstacle to achieving our goals is the absence of a standard way to request metadata information about files in a smartcard. Some information essential for the UNIX filesystem is simply not present in a smartcard, *e.g.*, file sizes, directory contents, and time stamps. Without such information, it is impossible to implement the complete functionality of the UNIX filesystem. For example, without directory entries, it is impossible to implement `ls` properly.

We have two choices, with concomitant trade-offs:

- Dictate an internal format on a smartcard to store information such as directory entries, length of a file, *etc.*, in a file in a smartcard. This provides full functionality of UNIX filesystems.
- Degrade functionality of SCFS. For example, no `ls`, no `cat`.

We compromise between the two choices. We believe it is essential to be able to determine a smartcard's directory structure through UNIX commands such as `ls`, so SCFS requires directory structure information to be stored in a smartcard. We also require a smartcard to store file lengths because they are necessary to implement the `read` and `write` system calls. Every directory (or DF in ISO-7816) in a smartcard has a file called `2e.69` (".i") containing the requisite metadata.

2.3 Design

Inspired by Arla [19], SCFS is implemented as a kernel module, `xfs`, that handles VFS requests, and a user daemon, `scfsd`, that communicates with an ISO-7816 smartcard. Figure 1 shows the overview of the design.

When an application calls a VFS operation (*e.g.*, `read` or `write` to a smartcard file), the

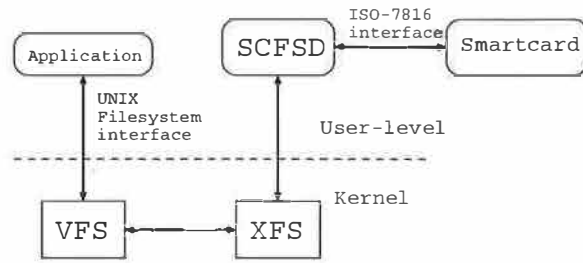


Figure 1: SCFS design

kernel module upcalls `scfsd` to request service. `Scfsd` creates ISO-7816 APDUs,⁵ sends them to a smartcard, gets returned data, and passes it to the kernel module.

Separation between `xfs` and `scfsd` allows us to use an existing ISO-7816 library [17] for handling the ISO-7816 protocol and dealing with its complex timing requirements. Kernel code is minimized, making SCFS easy to debug and port.

To absorb differences among smartcards, SCFS requires some knowledge of a smartcard before it is mounted, *e.g.*, existence of special APDUs, opcodes used for APDUs, ATRs⁶ they return, *etc.* The information is stored in a SCFS configuration file, `/usr/scfs/etc/scfs.scdb` by default.

SCFS automatically identifies a smartcard type from its ATR. When a reset signal is sent to a smartcard, it responds with a 4 - 32 byte ATR, unique to each smartcard type. The SCFS configuration file has a database of known ATRs. If the ATR from the smartcard is listed in the configuration file, SCFS retrieves the entry for that type of smartcard. Details about the configuration file are described in Section 3.6.

Unlike most UNIX filesystems, SCFS does not cache data read or written because caching might degrade the security of data resident in a smartcard. Data in the UNIX filesystem (typically a hard disk) is not protected as securely as

⁵An *Application Protocol Data Unit*, or APDU, can be viewed as a framing protocol for messages passed from application software to a smartcard [9].

⁶Answer To Reset.

in a smartcard and is not protected at all from an adversary with administrative privileges. In addition, files in a hard disk are usually backed up on tape, which may fall into the hands of an adversary.

3 Implementation

3.1 Overview

As described in Section 2.3, SCFS is separated into a kernel module (**xfs**) and a SCFS daemon (**scfsd**), detailed in Sections 3.2 and 3.3, respectively. Communication between **xfs** and **scfsd** is detailed in Section 3.4. Implementation of SCFS is based on Arla-0.6. Communication between **xfs** and **scfsd** is derived directly from Arla.

3.2 Kernel Module (xfs)

The kernel module (**xfs**) implements a virtual filesystem, the `ioctl` system call, and communication with **scfsd**.

The virtual filesystem consists of several functions called by the kernel when a file in SCFS is accessed. For example, the core part of the `read` system call is implemented by the `xfs_read()` vnode operation in the **xfs**.

We describe some important vfs operations, `xfs_mount()` and `xfs_root()`, and some important vnode operations, *i.e.*, `xfs_lookup()`, `xfs_read()`, `xfs_write()`, `xfs_getattr()`, and `xfs_readdir()`, in Section 3.5.

Xfs is typically loaded into the kernel at boot time. When **xfs** needs to communicate with a smartcard, it performs the communication by upcalling **scfsd**. For example, `xfs_read()` invokes `xfs_message_readsc()` in **scfsd**. **Xfs** waits until it receives data from **scfsd**, and sends the data back to the application with the `uiomove` kernel function.

3.3 SCFS daemon (scfsd)

Scfsd performs operations requested by **xfs**. For requests that require smartcard access, **scfsd** translates the request to ISO-7816 AP-DUs. Figure 2 shows an example of message flow when an application requests to read 8 data bytes from a smartcard.

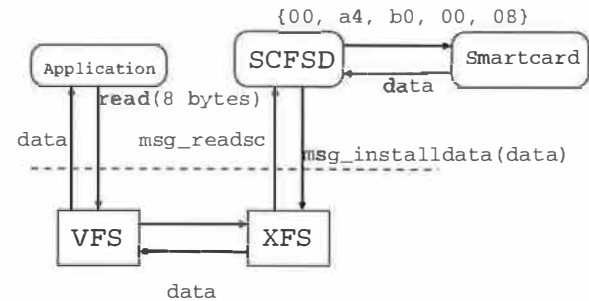


Figure 2: Reading 8 data bytes from a smartcard

3.4 Communication between xfs and scfsd

Xfs communicates with **scfsd** through RPC. When **xfs** needs access to a smartcard, it constructs a request message, puts it into a message queue, and waits for **scfsd** to reply. Code for sending a request to read 8 bytes from a smartcard is as follows:

```

struct xfs_message_readsc msg;

msg.header.opcode = XFS_MSG_READSC;
msg.buf = buf;
msg.size = 8;
msg.offset = 0;
fidcpy(msg.fid, xnode->handle);
xfs_message_rpc(fd, &msg.header,
                sizeof(msg));
  
```

After invoking `xfs_message_rpc()`, **xfs** sleeps until it receives the result of the request. **Scfsd** eventually receives data from a smartcard and sends it back to the kernel module. Here is an example of sending a reply message:

```

struct xfs_message_installdata msg;

msg.header.opcode = XFS_MSG_INSTALLDATA;
memcpy (msg.buf, data);
msg.size = size;
xfs_send_message_wakeup(fd, error, msg);

```

3.5 Important VFS/Vnode operations

In this section, we detail the implementation of some important VFS and vnode operations.

VFS Operations:

- **Xfs_mount()** mounts SCFS on a specified directory. It first sends a reset signal to the smartcard. When it receives ATR from the smartcard, it scans the configuration file to find a smartcard description that matches the ATR, reads the configuration information, initializes **scfsd**, initializes **xfs**, and creates the mount point.
- **Xfs_root()** operation selects a root directory (3f.00) in a smartcard and installs an XFS node and a **vnode** for a root node.

Vnode operations:

- **Xfs_lookup()** translates a path to an 8 byte **fid**.⁷ It checks if the requested path-name and its parent are both in the directory structure. If they are, it constructs and returns the **fid**. Currently, a path length is restricted to four components because a **fid** is 8 bytes long, big enough to hold four ISO-7816 components, which are two bytes each. We map these two bytes into their ASCII equivalents in the natural way.
- **Xfs.read()** reads data from a (possibly PIN-protected) smartcard file, as follows.

⁷A **fid** is a file identifier that is unique in SCFS, consisting of names of the file itself and its ancestors. For example, a **fid** of a file 3f.00/77.77/77.01 is 77.01.77.77.3f.00.ff.ff.

(1) It selects the target file. (2) When the current file and the target file have the same parent, the target file is selected by a select APDU. Otherwise, the entire path from the root must be navigated; ISO-7816 does not allow selection of an arbitrary file, only one in the currently selected directory, so in this case, **xfs.read()** selects the root file (3f.00), and moves down a path one by one to the target file. (3) With the file now selected, **xfs.read()** sends a read APDU (e.g., c0 b0 00 00 length) to the smartcard. (4) If the read request fails because the file is protected by a PIN, **scfsd** prompts the user for a PIN. The prompt is directed to the controlling tty of the application that issued the system call. (5) Finally, **scfsd** passes the data read back to the user via a call to the **xfs** layer and kernel **uiomove()**.

- **Xfs.write()** behaves identically to **xfs.read()**, except for the direction of data.
- **Xfs.getattr()** installs a VFS attribute structure (**struct vattr**) and an XFS attribute structure (**struct xfs.attr**). **Scfsd** performs the actual construction of the XFS attribute structure and sends it to **xfs**, which converts it into a VFS attribute structure.
- **Xfs.readdir()** is typically called by a **getdirenties()** system call, often as a result of an **ls** command. It returns directory entries (**struct dirent**) of a selected directory. Each entry describes a file or a directory in the selected directory. ISO-7816 shortcomings require that we define our own metadata strategy, described in Section 2.2. **Xfs.readdir()** constructs full directory entries from the directory entries and from our metadata file and returns them to the application.

Some functionalities in a smartcard do not fit the concept of a filesystem. For example, there is no system call to read a PIN to authorize a user. However, these functionalities are necessary to take advantage of security features of

a smartcard. To incorporate them into SCFS, we use the `ioctl()` operation.⁸ `ioctl()` takes an opcode and data and performs an opcode-specific action.

Implementation of `ioctl()` is straightforward, translating one opcode to one APDU. `ioctl()` implements create file, verify PIN, verify a key, internal authentication, external authentication, get response, and get challenge APDUs.

3.6 Configuration File

The configuration file (stored in `/usr/scfs/lib/scfs.scdb` by default) includes entries for ATR, the name of the smartcard, the CLA byte used for APDUs, whether the APDUs are supported by the smartcard, the type of supported PIN protection, *etc.* An example of a configuration file is as follows:

```

ATR    3b 32 15 0 49 10 {
    CARDNAME           CyberFlex
    MULTIFLEXPIN        no
    MULTIFLEXGETRES     no
    CLA_DEFAULT         c0
    CLA_VERIFYKEY       f0
    CLA_READBINARY      f0
    CLA_UPDATEBINARY    f0
    CLA_READRECORD      -1
    CLA_UPDATERECORD    -1
}

ATR    3b 2 14 50 {
    CARDNAME           MultiFlex
    MULTIFLEXPIN        yes
    MULTIFLEXGETRES     yes
    CLA_DEFAULT         c0
    CLA_VERIFYKEY       f0
}

ATR    3b 23 0 35 11 80 {
    CARDNAME           PayFlex/MCard
    MULTIFLEXPIN        no
    MULTIFLEXGETRES     no
    CLA_DEFAULT         00
}

```

⁸ We use `ioctl()` to avoid adding a new system call; this decision will be revisited someday.

The byte string after the “ATR” tag is matched with the ATR returned from a smartcard at reset. The `CLA_*` tags defines CLA bytes for specific APDUs, used by `scfsd` to construct APDUs. -1 means that the APDU is not supported in the smartcard type. If a CLA byte is not specified for the APDU, `CLA_DEFAULT` is used. For example, in CyberFlex, the CLA byte is `0xf0` for the `verify_key`, `read_binary`, and `update_binary` APDUs. `Read_record` and `update_record` APDUs are not defined. `0xc0` is used for the CLA byte for the other APDUs.

4 Performance Evaluation

Here we evaluate the performance of SCFS, implemented on two Schlumberger cards, MultiFlex and CyberFlex Access. Our test harness is based on a 400 MHz Pentium running OpenBSD-2.4.

4.1 Method

We measured total elapsed time and smartcard access time for various vnode operations. The difference reflects filesystem overhead. Figure 3 shows this relation.

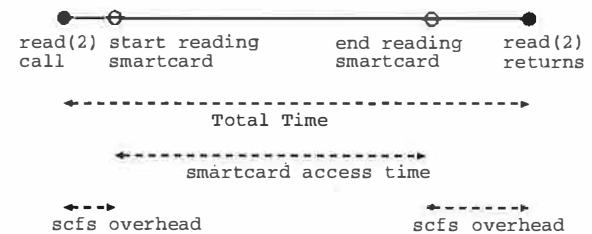


Figure 3: Performance Evaluation

Serial communication with smartcards uses 12 bits per byte (one start, eight data, one parity, two stop bits). Our test harness communicates with MultiFlex at 38.488 Kbps, or 312 μ sec. per byte, and with CyberFlex Access at 55.928 Kbps, or 215 μ sec. per byte.

4.2 Result

To measure read and write performance, we used six different operand sizes, ranging from 1 byte to 254 bytes. Figure 4 shows the result. For both cards, elapsed time as a function of operand size is very close to linear.

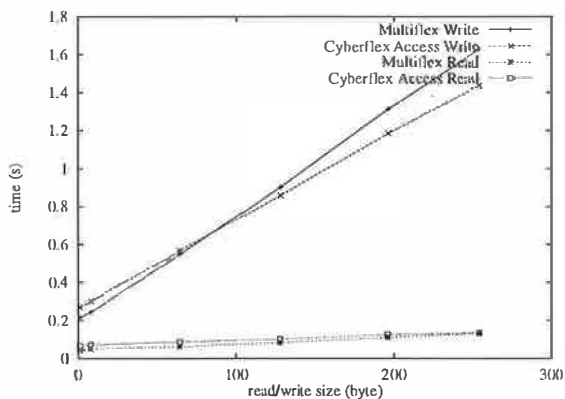


Figure 4: read/write time

Table 1 shows the card-related per-byte cost for these functions. Most of the overhead, evident by the non-zero y-intercept, is due to (unknown) card processing time; operating system overhead is under 1 ms in each case. Some of the per-byte card processing time is evident in the difference between the theoretical minimum read cost and the measured cost. Write times are substantially longer than the theoretical minimum, reflecting the time required to write to EEPROM.

Card	Op	Per-byte
MF	read	0.345
CFA	read	0.257
MF	write	5.62
CFA	write	4.62

Table 1: Read and write performance. All times in ms.

4.3 Breakdown

Table 2 shows the cost of some other operations for the CyberFlex Access card.

Op	Card	OS	Bytes
open	0	.566	0
lseek	0	0.339	0
create	466	2.00	78
remove	626	37.4	64
vrfykey	258	1.67	10

Table 2: Read and write performance. Times in ms. Bytes represent the amount of data transferred in the operation.

The directory structure cache, created at mount time by reading the “.i” file, is evident in the open and lseek operations, which do not communicate with the card. Verify key, not a vnode operation, is implemented with pioctl.

5 Discussion

5.1 Related Work

Here we discuss three important related works, OCF, PC/SC, and some special purpose standards.

OCF

OpenCard Framework is middleware that supports a smartcard with Java [2, 8] by providing high-level APIs, vendor transparency, card type transparency, and extensibility. These objectives are similar to ours. The principal advantage of OCF is that it employs Java. Programmers familiar with Java can start smartcard programming easily. The following is an example taken from “OpenCard Framework 1.1 Programmer’s Guide” [7]. It reads a file “id” (0x6964) and prints it out to standard output.

```

public static void main(String[] args)
{
    System.out.println(
        "reading smartcard file...");

    try {
        SmartCard.start();

        // wait for a smartcard with file
        // access support
        CardRequest cr =
            new CardRequest(
                FileAccessCardService.class);
        SmartCard sc =
            SmartCard.waitForCard(cr);

        FileAccessCardService facs =
            (FileAccessCardService)
            sc.getCardService(
                FileAccessCardService.class, true);
        CardFile root = new CardFile(facs);
        CardFile file =
            new CardFile(root, ":6964");

        byte[] data =
            facs.read(file.getPath(), 0,
                file.getLength() );
        sc.close();

        String entry = new String(data);
        entry = entry.trim();
        System.out.println(entry);

    } catch (Exception e) {
        e.printStackTrace(System.err);
    } finally { // even in case of an error
        try {
            SmartCard.shutdown();
        } catch (Exception e) {
            e.printStackTrace(System.err);
        }
    }

    System.exit(0);
}

```

The example code is easy to understand for those familiar with Java. Programmers can take advantage of the higher abstraction of

Java, such as I/O streams, *etc.* OCF is integrated with JavaCards, providing a consistent development environment for application software and on-chip software.

However, the reliance on Java can also be viewed as a disadvantage. Java and its object oriented model modularize and simplify complex software, but a smartcard is a simple, passive device. For many smartcard applications, Java might be viewed as overkill.

In SCFS, we use a smartcard in a simple way. For example, we can print out a file (as in the OCF example) by typing:

```

% mount_scfs /dev/scfs0 /smartcard
% cat /smartcard/id

```

OCF cannot be used with languages such as C and C++, the languages in which most operating systems and security protocols are written. Consequently, OCF offers little to enhance directly the security of many operating systems and security protocols, such as UNIX, Kerberos, SSH, and PGP.

PC/SC

PC/SC is a general purpose architecture for integrating a smartcard into PCs [3]. Its objectives are similar to OCF and SCFS. According to part 6 of the specification [4], the PC/SC API is similar to the UNIX filesystem, featuring `Open()`, `Close()`, `Read()`, `Write()`, `Seek()`, *etc.* Therefore, usability of PC/SC and SCFS are similar.

Unlike OCF, PC/SC supports multiple languages and development environments, such as C, VC++, VB++, and Java. However, it is used only with Windows operating systems. While SCFS currently supports only OpenBSD, it is possible to port it to other UNIX systems, and (perhaps) even to Windows NT.⁹

⁹If we purchase the Installable Filesystem package.

Special Purpose Standards

Application specific standards such as PKCS#11, EMV, and SET have advantages in usability in specific domains because of higher abstractions than SCFS. In SCFS, functionality to take advantage of smartcard security, such as internal and external authentication, is given by the `ioctl()` system call. However, `ioctl()` is not as user friendly as the functionality provided by PKCS#11, EMV, and so on. We may provide libraries for specific purposes to wrap around SCFS to give higher abstractions.

5.2 Advantage of SCFS

Transparent API with the UNIX Filesystem

SCFS differs from the other approaches such as OCF and PC/SC because it is implemented as an operating system extension. Consequently, to an application, smartcard files look identical to files stored on other media. With SCFS, an application can use a smartcard without modification (Figure 5).

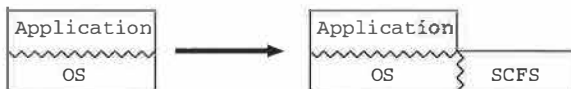


Figure 5: Application is not modified to use SCFS.

With SCFS, many UNIX applications can take advantage of smartcard security without modification. For example, here is how we made SSH work with a private key stored in a smartcard: we added a symbolic link from `$HOME/.ssh/identity` to `/smartcard/ss/id` and copied a private-key to the SSH identity file.

```
citi% mount_scfs /dev/scfs0 /smartcard
citi% ln -s /smartcard/ss/id
        ~/.ssh/identity
citi% ssh sin.citi.umich.edu
```

```
Enter PIN:
sin% logout
```

PGP works with a private key in a smartcard in a similar way:

```
citi% mv ~/.pgp/secring.pgp
        /smartcard/pg/ky
citi% ln -s /smartcard/pg/ky
        ~/.pgp/secring.pgp
```

Although not tested yet, Kerberos tickets and browser cookies can be stored in SCFS in similar ways.

In contrast, OCF or PC/SC require that an application be modified to use a smartcard because the API for a smartcard is different from the API for normal files (Figure 6).

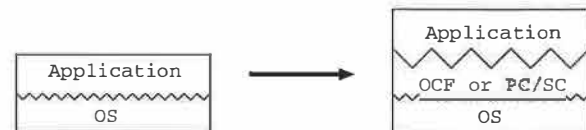


Figure 6: Application must be modified to use OCF or PC/SC

Portability

Another advantage of SCFS is portability. Most of the SCFS code is in user space and easily ported to other operating systems. The `xfs` kernel module is based on `Arla`, which is already ported to many UNIX-like operating systems, including Solaris, NetBSD, FreeBSD, OpenBSD, Linux, AIX, HP-UX and Digital UNIX. It is easy to port SCFS `xfs` to other operating systems.

5.3 SCFS as Development Tool

Smartcard standards other than SCFS give higher abstractions for users, *e.g.* Java language in OCF, EMV'96 for electric commerce,

PKCS#11 for cryptographic applications, *etc.* Depending on the type of applications, different kinds of abstraction may be required. Therefore, there are many standards that do not interoperate [1]. In contrast, SCFS works with a raw smartcard with a minimum amount of abstraction; no matter what functionality a smartcard offers, SCFS can access and use its secure storage. SCFS allows users to access a smartcard with sophisticated UNIX commands, such as `cd`, `ls`, `pwd`, `cat`, *etc.* SCFS is especially helpful in maintenance, testing, and debugging; Figure 7 depicts our model of SCFS as a development tool.

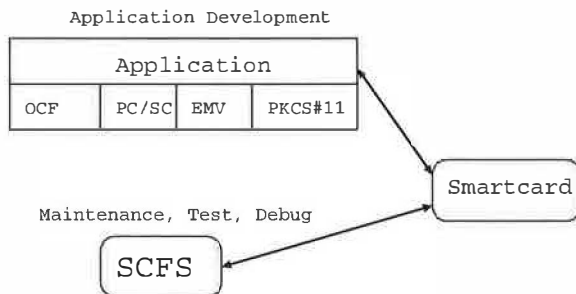


Figure 7: SCFS as a low-level development tool.

6 Future Directions

Some ideas derived from other smartcard standards suggest enhancements to SCFS. In PC/SC, a smartcard specific driver is loaded as a DLL (Dynamic Loadable Library). In SCFS, smartcard specific code is directly written in the user-level daemon, `scfsd`. PC/SC's approach is more extensible than ours because it does not require recompilation to add a driver for a new smartcard. We are considering extending SCFS to have the same advantage with dynamically loadable libraries.

We intend to port SCFS to different operating systems and to support more smartcard types. (Currently it supports only Schlumberger MultiFlex, CyberFlex, and PayFlex).

Security of SCFS should be explored. SCFS is

currently vulnerable to Trojan Horse attacks, *i.e.*, if an adversary has administrative privileges, she can install a rogue version of SCFS that steals a user's PIN or modifies contents of the smartcard. We are investigating integrity checking and authentication of SCFS code by a smartcard.

We plan to use SCFS in several applications. One of them, storing Kerberos tickets, is particularly interesting, as it dovetails with our related Kerberos V5 smartcard extensions [10]. In that application, the smartcard performs decryption on Kerberos tickets. Storing the result in a protected SCFS file indicates the synergy of our approach.

7 Conclusion

We have implemented a Smartcard Filesystem (SCFS) to ease development of smartcard software. SCFS provides a UNIX filesystem API for a smartcard. Developers can use the well-established UNIX API and development environment to develop smartcard software. Performance evaluation shows the overhead caused by SCFS is negligible.

8 Acknowledgment

We thank the Arla developers: Assar Westerlund, Love Hornquist-Astrand, Magnus Holmberg and many more people in the arla-drinkers mailing list for patiently answering our questions.

This work was supported by Schlumberger's Program in Smartcard Technology at CITI.

References

- [1] Duncan W. Brown. Application development: A new focus for smart

- card suppliers and implementers. In *CardTech/SecureTech'98*, volume 1, pages 352–353, Washington, DC, April 1998.
- [2] OpenCard Consortium. General information web document, Oct. 1998. <http://www.opencard.org/docs/gim/ocfgim.html>.
- [3] Microsoft Corporation. Smart cards, white paper, April 1998. <http://www.microsoft.com/smartcard/smartcards/scardwp.asp>.
- [4] PC/SC Workgroup (Microsoft Corp. etc.). Interoperability specification for ICCs and personal computer systems, part 1-8, December 1997. <http://www.smartcardsys.com>.
- [5] Europay, MasterCard, and Visa. EMV'96: Integrated circuit card application specification for payment systems, June 1996. <http://www.mastercard.com/emv/emvspecs02.html>.
- [6] Scott B. Guthery and Timothy M. Jurgensen. *Smart Card Developer's Kit*. MacMillan Technical Publishing, Indianapolis, Indiana, December 1997.
- [7] R. Hermann, D. Husemann, and P. Trommler. OpenCard framework 1.1 programmer's guide, Oct 1998. <http://www.opencard.org/docs/pguide/PGuide.html>.
- [8] Reto Hermann, Dick Huseman, and Peter Trommler. The OpenCard framework. In *CARDIS'98*, Louvain-la-Neuve, Belgium, Sept. 1998. Third Smart Card Research and Advanced Application Conference.
- [9] The International Organization for Standardization and The International Electrotechnical Commission. *ISO/IEC 7816-4 : Information technology - Identification cards - Integrated circuit(s) cards with contacts*, 9 1995.
- [10] Naomaru Itoi and Peter Honeyman. Smartcard integration with Kerberos V5. In *Proceedings of USENIX Workshop on Smartcard Technology*, Chicago, May 1999.
- [11] S. R. Kleiman. Vnodes: An architecture for multiple file system types in sun unix. In *Proceedings of USENIX Summer Technical Conference*. USENIX, 1986.
- [12] RSA Laboratories. PKCS #11: Cryptographic token interface standard. version 2.01, December 1997. <http://www.rsa.com/rsalabs/pubs/PKCS/>.
- [13] SET Secure Electronic Transaction LLC. SET standard technical specifications, 1999. <http://www.setco.org/>.
- [14] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley Publishing Company, 1996.
- [15] SUN Microsystems. Java card technology. <http://java.sun.com:80/products/javacard/index.html>.
- [16] MULTOS. <http://www.multos.com/>.
- [17] Jim Rees. ISO 7816 library, 1997. <http://www.citi.umich.edu/projects/sinciti/smartcard/sc7816.html>.
- [18] James F. Russell. Compatibility and conflicts: PC/SC, OCF, Java card, MULTOS ... In *CardTech/SecureTech'98*, volume 1, pages 97–101, Washington, DC, April 1998.
- [19] Assar Westerlund and Johan Danielsson. Arla - a free AFS client. In *Proceedings of USENIX 1998 Annual Technical Conference*, pages pp. 149 – 152, New Orleans, Louisiana, USA, June 1998. USENIX. <http://www.stacken.kth.se/projekt/arla>.

Secure Object Sharing in Java Card

Michael Montgomery
Austin Product Center
Schlumberger
Austin, TX 78726
mmontgomery@slb.com

Ksheerabdh Krishna
Austin Product Center
Schlumberger
Austin, TX 78726
kkrishna@slb.com

Abstract

Since the invention of the Java Card, the issue of code and data sharing has been a topic of great interest. Early Java Cards shared data via files secured with access control lists. Java Card 2.1 specification introduced a method of object sharing, allowing access to methods of server applets using Shareable Interface Objects (SIO). However, this SIO approach can be improved. It permits access to all interfaces of the SIO, whereas some interfaces may be intended only for particular clients. AID impersonation could be used to gain access to services unless the card authenticates all applets. Access to a SIO by future applets may be impossible. Passing object data between applets is quite cumbersome.

An approach to object sharing based on delegates is described, which provides needed improvements with minimal modifications to Java Card 2.1. Using the delegate approach, only the desired methods of an applet are exposed, and each method can be protected by any security policy the applet wishes to implement. A shared secret security policy is described, using challenge/response phrases to avoid revealing the shared secret. Such a security policy does not require applet authentication to avoid AID impersonation, and lends itself readily to access by any future applets that may be written.

1 Introduction

Since the invention of the Java Card, the issue of code and data sharing has been a topic of considerable interest. The first Java Cards [2] shared data between Java Card applets using a file system secured by access control lists. These lists determined which identities could access particular files, and what permission each identity was granted with respect to each file. The identities were established using key files and PIN verification. This solution was quite powerful for many common data sharing situations; however it did not lend itself well to situations requiring access to methods belonging to another Java Card applet.

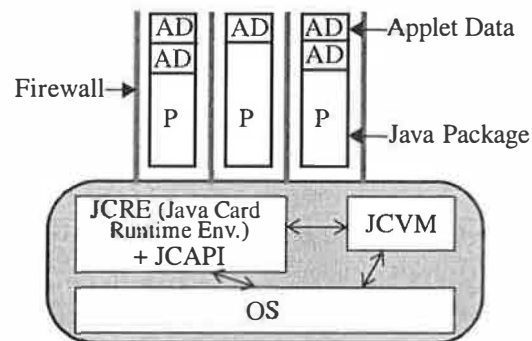


Figure 1: Basic Java Card Architecture

A typical Java Card architecture is shown in Figure 1. The card contains an operating system, Java Card Virtual Machine [7], Java Card Runtime Environment (JCRE), and the Java Card API in ROM. Applications are Java packages programmed to the API, and are usually loaded into EEPROM. An application consists of one or more applets; applets in different packages are separated by a firewall to prevent access to applet data across package boundaries. Applets are programmed using a subset of Java, and have a specified set of entry points that trigger various actions on the card [6].

In this paper, we will describe the object sharing mechanism introduced in Java Card 2.1, examine the issues associated with this mechanism, and propose alternatives that address these issues. We rely heavily on code examples, key elements of which are inserted into the paper at relevant points. Sometimes the code in the paper is edited for brevity, and comments containing "..." indicates the removal of code not pertinent to the discussion. The complete Java source code for these examples can be found in <http://www.cyberflex.slb.com/usenixMK99.html>. Note that when discussing the delegate examples, this code uses a framework which is suitable for simulation on a workstation, and is not intended for use on a Java card.

2 Object Sharing In Java Card 2.1

The Java Card 2.1 specification [10] introduces a means of sharing objects between Java Card applets. Although somewhat more difficult to use than file sharing, it does provide a means for accessing object methods, rather than just data. This specification uses a unique applet identifier (AID) [4] as the basis for determining which applets are granted access to objects created by other applets.

2.1 Description Of Object Sharing Mechanism

The strict firewall enforcement of Java Card 2.1 completely prevents an applet from accessing data corresponding to another applet. However, a provision was made for an applet to obtain an interface belonging to another applet, and to invoke a method on this interface. This forms the basis of the Java Card 2.1 object sharing mechanism.

2.1.1 Restricting Method Access through a SIO

Normally in Java [1], it would be possible to access public methods across packages. Therefore, one could use public methods in other packages without a need for a sharing mechanism.

But this poses a problem for Java card, because the applet entry points are necessarily public, yet we must restrict access to them to prevent applets from running other applet methods without permission. So the JCRE

does not permit any method to be invoked in other applets, except through the SIO mechanism. This prevents obvious hacks, such as casting an object reference to gain access to all of the public object methods, bypassing the restriction of the interface.

2.1.2 Applet Context

The object system in a Java Card is partitioned into separate protected object spaces referred to as contexts. All applets in a given package share the same context, and are prohibited from accessing objects in a different context due to firewalls which are enforced by the Java Card Runtime Environment (JCRE) [8]. The JCRE is able to access objects in any context, and global arrays such as the APDU buffer (which are owned by the JCRE) can be accessed by applets in any context.

2.1.3 Shareable Interface Objects (SIO)

Shareable interfaces are a new feature in the Java Card 2.1 API [9] to enable applets to explicitly share objects by defining a set of shared interface methods. Such shareable objects are called Shareable Interface Objects (SIO). We can think of those applets which provide SIO as server applets (since they will provide access to their services via the SIO), and those applets which use the SIO of another applet as client applets. Note that an applet may be a server to some applets, and yet a client of other applets.

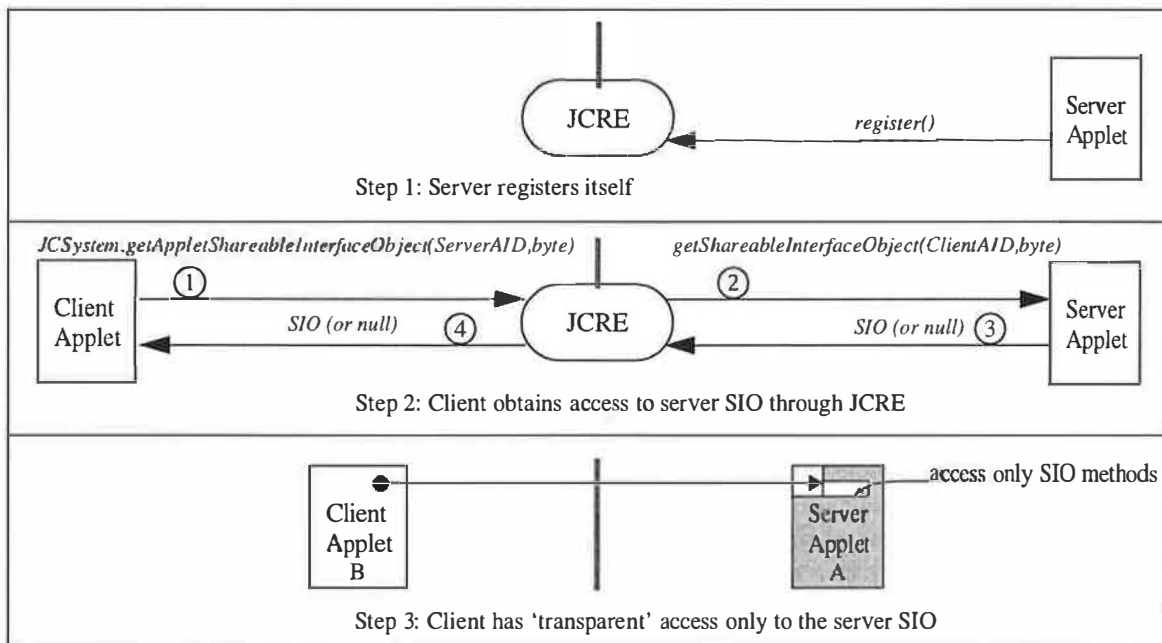


Figure 2: Creating and accessing a SIO

2.1.4 Creating a SIO

In order to create a new SIO, a server applet A must first define a shareable interface X, which extends the interface `javacard.framework.Shareable`. Applet A then defines a class C that implements the shareable interface X, and creates an instance O of class C. Object O is now a SIO. An instance of applet A with an AID is registered with the JCRE, as shown in Figure 2, step 1; this AID is subsequently used by client applications to specify the server applet.

2.1.5 Obtaining a SIO

A client applet must obtain this SIO in order to access object O. The process of a client obtaining an SIO is shown in Figure 2, step 2.

In order for client applet B to access object O, applet B must create an object reference BO of type X, and then call a system method `getAppletShareableInterfaceObject` with the AID of the server, and an optional byte which selects which interface is desired (for those servers with more than one interface available). The JCRE looks up the server applet associated with that AID, and forwards the request to the server, replacing the first argument with the client AID. Server applet A receives the request and the AID of the requester (applet B), and determines whether or not it will share object O with applet B. If applet A finds the request agreeable, then a reference to O is provided; otherwise, null is returned. The JCRE forwards this reference to Applet B. Applet B receives this reference (which is of type SIO), casts it to type X, and stores it in BO.

2.1.6 Using a SIO

Once a SIO has been obtained, applet B can invoke any methods from interface X on object reference BO, which then accesses object O. However, this is not as straightforward as it might seem, due to the firewalls.

Figure 2, step 3 shows the client applet B on the left, and the server applet A on the right, with a firewall in between. Note that the only part of applet A that is visible through the firewall is object O. When applet B invokes a method on BO, a context switch is triggered in the JCRE, leading to the situation in Figure 3. At this point, applet B is not visible at all; the only data visible from B are the arguments passed on the stack (and the global APDU array).

Note that it is pointless to pass object references on the stack, since the firewall will prevent object O (in Applet A) from using them, due to the context switch.

However, object O can access any allowed data in Applet A as shown in Figure 3.



Figure 3: Server has no access to client data

2.2 Object Sharing Issues

There are four issues of concern with object sharing in Java Card 2.1.

2.2.1 Access To All Interface Methods Of A Class

The JCRE protection mechanisms do not prevent interfaces from being maliciously cast into other kinds of interfaces that might exist for a given object.

Once granted any interface, it is possible to access all of the shareable interface methods of an object via an explicit cast (not just the interface granted to a particular client). So if my server applet has two interfaces, intended for different kinds of clients, a client who legitimately obtains one interface, could cast it into the other interface, and gain access to unintended methods.

For example, suppose an applet `ABCLoyaltyApplet` has two different services that it intends to offer exclusively to two different clients, i.e. `grantFrequentFlyerPoints` for client `Airline`, and `grantLoyaltyPoints` for client `Purse`. Both of these services are accessible to respective clients via different published interfaces, but both are defined in the Applet class, as illustrated in Code 1.

```
package AppABC;
import javacard.framework.*;

public interface ABCLoyaltyAirlineInterface extends Shareable {
    public void grantFrequentFlyerPoints(int amount);
}

public interface ABCLoyaltyPurseInterface extends Shareable {
    public void grantLoyaltyPoints(int amount);
}

public class ABCLoyaltyApplet extends
    javacard.framework.Applet implements
    ABCLoyaltyAirlineInterface, ABCLoyaltyPurseInterface {

    protected int loyaltyPoints;
    protected int frequentFlyerPoints;

    /* A service only for client 'Airline' */
    public void grantFrequentFlyerPoints(int amount) {
        frequentFlyerPoints += amount;
    }

    /* A service only for client 'Purse' */
    public void grantLoyaltyPoints(int amount) {
        loyaltyPoints += amount;
    }

    public ABCLoyaltyApplet() throws Exception {
```

```

        loyaltyPoints = 0;
        frequentFlyerPoints = 0;
        /* Add code to initialize potential client AIDs ... */
        register();
    }

    public void process() {
        /* Various code specific to services for this applet,
        such as point redemption code ... */
    }

    public Shareable getShareableInterfaceObject(
        AID clientAID, byte parameter) {
        if (clientAID.equals(purseAppletAID))
            return (ABCLoyaltyPurseInterface) this;
        else if (clientAID.equals(airlineAppletAID))
            return (ABCLoyaltyAirlineInterface) this;
        else
            return null;
    }
}

```

Code 1: Server defines and grants access to SIO

If the Purse client was aware of the Airline client interface, it could make use of the interface it had been granted by the server applet to grant loyalty points, and instead use the grantFrequentFlyerPoints interface (intended for client Airline) to maliciously add unwarranted frequent flier points, as shown in Code 2.

```

package AppPurse;

import javacard.framework.*;
import AppABC.*;

public class PurseApplet extends javacard.framework.Applet {

    private int value;
    private ABCLoyaltyPurseInterface abcSIO;

    public PurseApplet() throws Exception {
        value = 0;
        abcSIO = (ABCLoyaltyPurseInterface)
            JCSysm.getAppletSharableInterfaceObject(
                abcLoyaltyAppletAID, (byte)0);
        register();
    }

    public void use(int amount) {
        value -= amount;
        if (abcSIO != null)
            abcSIO.grantLoyaltyPoints(amount);

        /* Attempt to 'hack' by using an SIO to access the Airline
        * applet's exclusive service grantFrequentFlyerPoints */
        ABCLoyaltyAirlineInterface hackSIO =
            (ABCLoyaltyAirlineInterface)
            JCSysm.getAppletSharableInterfaceObject(
                ABCLoyaltyAppletAID, (byte)0);
        if (hackSIO != null)
            hackSIO.grantFrequentFlyerPoints(amount);
    }

    /* Other methods follow for adding points to the purse ... */
}

```

Code 2: Client compromises SIO

Unfortunately, the JCRE is unable to prevent such access, since it does not violate the restriction of access only via shareable interfaces.

This problem can be eliminated by using separate delegate objects for each shared interface, and have the delegate objects handle or redirect the calls to the intended object. Another solution is to verify the AID of the caller upon each attempt to access a method in the server.

Furthermore, it is not possible to grant a client access to only certain methods of an interface. If such

granularity is required, further delegates and interfaces must be used to separate the particular methods that are to be allowed for each applet. Alternatively, this granularity can be provided by having each method individually check the client AID, to determine whether the client should have access to that particular method.

2.2.2 AID Impersonation

The decision by a server applet to grant access to an object must be based solely on the AID of the requesting applet; no other information is available to the server applet. The intended use is clearly to allow certain interfaces to be granted only to particular client applets, as denoted by their unique AIDs. However, it is possible to maliciously set the AID of a rogue applet to be the same as the AID of a client applet known to have access to a particular interface. The rogue applet is then loaded *instead* of the applet that legitimately owns the AID. Once loaded, the rogue applet can request the desired interface. The server applet, having only the AID for reference, will naturally grant this request, since the AID matches the required AID for the interface. The rogue applet can then freely use the interface for malicious purposes.

This is a critical security problem. Current solutions to this problem require restrictions on applet loading, such as only allowing applets to be loaded that are signed by trusted sources. However, this approach greatly reduces the flexibility of Java Cards; for example, this prevents users from loading Java Card programs of their own devising.

2.2.3 Future Reference To Shared Objects

Granting access by AID necessarily assumes that the server applet that wishes to share the object has foreknowledge of all AIDs of all client applets that are to be loaded which will share particular interfaces. This is a difficult requirement for any kind of server. But what about other client applets that are written afterwards which legitimately need access to the shared object? Such applets are excluded from access to the object, since the server can only grant access based on the AID list that the server had when it was loaded. This necessitates rewriting and reissuing the server applet such that the new AIDs are included, which may pose an insurmountable hurdle when the server applet is already widely distributed.

2.2.4 Inability To Pass Object Parameters

The inability to pass object parameters between applets can make many tasks cumbersome. For example, supposes as a client needs to pass an object when invoking

ing a method on a server applet so that the server can manipulate and return the object, as in encrypting a buffer. In this case, the object might contain a field specifying the method of encryption, a key array, and a data array. But as stated earlier, the firewall prevents the server applet from accessing this data, even if an object reference from the client is provided.

One might envision a work around to this problem where the server A gets a shared interface on an object in client B in order to read the object data as shown in Figure 4.

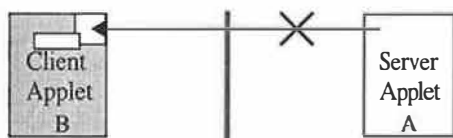


Figure 4: Server access blocked by firewall

However, this will fail, since any attempt by server A to get the object data from client B will also be blocked by the firewall. The server can only invoke methods on the client object, but the client object has no means for returning the object data, other than a single field at a time. Thus the only current work around to this problem is to use the global APDU buffer to pass data as shown in Figure 5.

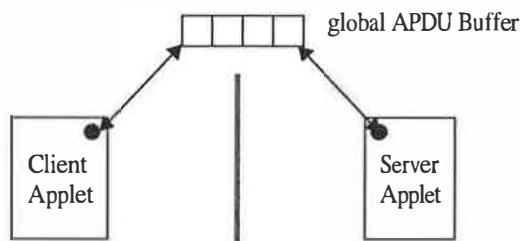


Figure 5: Data exchange via global APDU buffer

This is awkward at best, since the data being passed may not be of appropriate length or type to be readily passed via this mechanism. Consequently, the client and server might have to make multiple calls and considerable manipulations to pass the desired parameters, due to the restrictions of the APDU buffer.

This problem could be addressed by changing the firewall restrictions to permit access to shared objects (including data), instead of just interfaces; however, this could potentially create security holes. Applets must coded carefully to avoid exposing methods and data that are not intended to be shared. But if instead of actually sharing the applet object, the applet used a delegate to expose only the desired methods and data, the JCRE

specification could be altered to permit access to only delegate methods and data, thus eliminating the object parameter problem, with a minimum of security risk.

3 Delegate Approach To Object Sharing

Based on an analysis of these critical problems, an approach to object sharing was devised which avoids these drawbacks. In this approach, each server applet that wishes to permit access to its methods or data creates a single delegate, which is registered with the system based on the AID of the applet. The delegate exposes only those methods and data that the applet wishes to share. Access to the delegate is public; all methods in the delegate can be accessed by any other applet!

Since access to the delegate is unrestricted, an applet protects itself in two ways. First, the delegate is written to only access the desired methods of the server applet; other applet methods cannot be accessed. Second, the methods of the delegate can perform any checks as deemed necessary to check the validity of the access to the delegate; if the checks are not passed, the delegate can refuse to pass the request on to the applet.

3.1 Java Card 2.1 System Changes Required

This approach presumes the addition of two new Java Card 2.1 system methods. A version of register is needed which takes a delegate and AID as arguments. A system method `getDelegate` returns the delegate associated with a particular AID. These methods replace the `JCSys.getAppletShareableInterfaceObject` and `Applet.getShareableInterfaceObject` methods. Furthermore, the JCRE is altered to permit access to methods and data of delegate objects (DOs) in other contexts (just as it now permits access to methods of SIOs in different contexts).

For performance reasons, it would be worthwhile to investigate whether it is necessary for the JCRE to restrict access to objects in other contexts at all, since checking object context imposes a speed penalty. Techniques such as the delegate method proposed, coupled with classical Java language and runtime protections[11,5], could perhaps result in a better performing system that still meets the security requirements.

3.2 Delegate Server Implementation

A server applet must be written containing the desired methods. For this example, we show a loyalty server applet in Code 3 from the ABC company that allows granting, using, and reading loyalty points.

```

package AppABC;
import OSResources.*;

public class ABCLoyaltyApplet extends OSResources.Applet {
    private String aid = "AppABC.ABCLoyaltyApplet";
    private int loyalty;

    public ABCLoyaltyApplet() {
        loyalty = 0;
        register(new ABCLoyaltyDelegate(this), aid);
    }

    void grantPoints(int amount) {
        loyalty += amount;
    }

    boolean usePoints(int amount) {
        if (loyalty >= amount) {
            loyalty -= amount;
            return true;
        } else {
            return false;
        }
    }

    int readPoints() {
        return loyalty;
    }
}

```

Code 3: Server applet registering its delegate

Note that the only novel aspect of this applet is when it creates and registers its delegate, as illustrated in Figure 6, step 1. Whenever a server applet wishes to allow access to another applet, a delegate must be written which exposes the desired methods. Such a delegate is shown in Code 4.

```

package AppABC;
import OSResources.*;

public class ABCLoyaltyDelegate extends Delegate {

    final static byte CHALLENGE_LENGTH = (byte) 64;
    final static byte FAILURES_ALLOWED = (byte) 2;

    private byte[] secret1 = { /* initializing code ... */ };
    private byte[] secret2 = { /* initializing code ... */ };
}

```

```

private ABCLoyaltyApplet myApplet;
private ChallengePhrase cp;
private ChallengePhrase checkcp;
private byte grantTriesRemaining = FAILURES_ALLOWED;
private byte useTriesRemaining = FAILURES_ALLOWED;

protected ABCLoyaltyDelegate(ABCLoyaltyApplet a) {
    myApplet = a;
    cp = new ChallengePhrase(CHALLENGE_LENGTH);
    checkcp = new ChallengePhrase(CHALLENGE_LENGTH);
}

public boolean grantPoints(int amount) {
    // do some checking
    if (myApplet != null) {
        ABCLoyaltyInterface pD = (ABCLoyaltyInterface)
            OSSystem.getDelegate(
                OSSystem.getPreviousContextAID());
        if (pD != null) {
            /* Create new random challenge phrase */
            checkcp.setPhrase(cp.randomize());
            /* Get client response to phrase */
            byte[] response = pD.loyaltyChallenge(cp);
            byte[] r = checkcp.encrypt(secret1);
            if (isEqual(response,r)) {
                /* See if client gave the correct response */
                grantTriesRemaining = FAILURES_ALLOWED;
                myApplet.grantPoints(amount);
                return true;
            } else {
                if (--grantTriesRemaining == 0) myApplet = null;
                return false;
            }
        }
    }
    return false;
}

public boolean usePoints(int amount) {
    /* Access to this method uses secret2, for different
       clients, but is otherwise similar to grantPoints ... */
}

public int readPoints() {
    if (myApplet != null)
        return myApplet.readPoints();
    else
        return 0;
}
}

```

Code 4: Server delegate

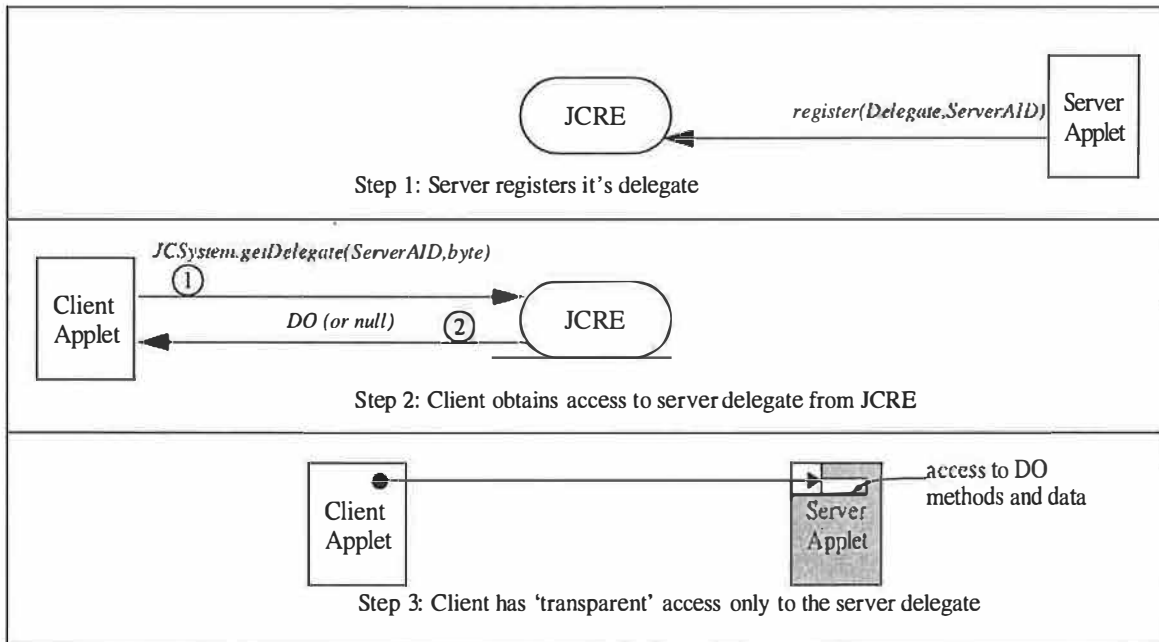


Figure 6: Creating and accessing a delegate

This delegate is registered with the AID of the server applet at the time the applet is instantiated on the Java Card. At this point, this delegate is publicly accessible by any other applet. This particular delegate defines the interface `ABCLoyaltyInterface` shown in Code 5, which must be implemented by client applications. The reasons for this interface are discussed in detail in Section 3.3.

```
package AppABC;
import OSResources.*;

public interface ABCLoyaltyInterface{
    public byte[] loyaltyChallenge(ChallengePhrase cp);
}
```

Code 5: Server defines interface that each client must provide to handle challenge/response

3.3 Delegate Security

A delegate controls which methods each client can access. This delegate exposes three methods: `grantPoints`, `usePoints`, and `readPoints`. The method `readPoints` has no validation, and may be freely used by any client.

But what about the methods that require proof that the access is valid, such as `grantPoints` and `usePoints`? These methods could just check the AID of the client. This effectively mimics the object sharing of Java Card 2.1, which is subject to the risks of AID impersonation and difficulty of adding future clients.

These problems can be avoided by using the classical solution of shared secrets. In this example, access to `grantPoints` is controlled by `secret1`; only clients that can authenticate knowledge of `secret1` will have their request passed from the delegate to the server applet. Access to `usePoints` is controlled by `secret2`, which allows a potentially different set of clients to access this method.

There are many standard techniques for handling shared secrets and other methods of authentication [3]. We are not proposing anything new here, other than the application of these techniques to this problem, to avoid the limitations and pitfalls of security based on AID. The details of the application of the shared secret technique to Java Card is presented in detail for the benefit of those who may not be familiar with such techniques, or may be unclear how they can be implemented in a Java card.

Proving knowledge of the secret could be a tricky proposition. Passing the secret as an argument to the server applet is a bad idea, since the secret could be intercepted in a number of ways. (One simple way to get the secret is to write an applet that impersonates the server applet, thus capturing the secret when it is passed as an argument.) A superior approach is to use challenge/

response phrases, as illustrated by the server delegate in Code 4. When access is requested, the server delegate sends a random challenge phrase to a predefined method in the client delegate. This challenge phrase is encrypted using the secret by the client delegate, and returned to the server delegate. The server delegate performs the same encryption, and if the results match, access is granted. Thus the secret is never revealed outside of either applet.

3.4 Delegate Client Implementation

When a client applet wishes to use the server applet, it calls `getDelegate` with the AID of the server applet, and receives a delegate, which it casts to the proper delegate class associated with the server applet, as shown in Figure 6, step 2. The client applet then invokes methods on the delegate as desired. Note that unlike SIO, the JCRE hands out the delegate, and server applet is not involved. The client in Code 6 was written assuming authentication using challenge/response.

```
package AppPurse;

import OSResources.*;
import AppABC.*;

public class PurseApplet extends OSResources.Applet {

    public String aid = "AppPurse.PurseApplet";
    private byte[] secret1 = { /* initializing code ... */ };
    private int value;

    public PurseApplet() {
        value = 0;
        register(new PurseDelegate(this), aid);
    }

    public byte[] loyaltyChallenge(ChallengePhrase cp) {
        return cp.encrypt(secret1);
    }

    public void use(int amount) {
        value -= amount;

        ABCLoyaltyDelegate d =
            (ABCLoyaltyDelegate)
            OSSystem.getDelegate("AppABC.ABCLoyaltyApplet");
        if (d != null) {
            d.grantPoints(amount);
        }
    }

    /* Other methods follow for adding points to the purse ... */
}
```

Code 6: Client applet invoking server method

To handle the challenge/response, the client must supply a delegate as shown in Code 7 that the server will call to perform the necessary encryption with the challenge phrase. This client delegate must implement the `ABCLoyaltyInterface`, as defined by the server applet. The delegate may not actually perform the encryption, but may instead pass the challenge/response request to the client applet for processing. This allows the delegate to avoid holding the shared secret, reducing the security risk.

```

package AppPurse;

import OSResources.*;
import AppABC.*;

public class PurseDelegate extends Delegate implements
ABCLoyaltyInterface{

    private PurseApplet myApplet;

    protected PurseDelegate(PurseApplet a) {
        myApplet = a;
    }

    public byte[] loyaltyChallenge(ChallengePhrase cp) {
        if (myApplet != null)
            return(myApplet.loyaltyChallenge(cp));
        else
            return null;
    }
}

```

Code 7: Client challenge/response delegate

3.5 Overview Of Client/Server Communication

At this point, it is presumed that the server has already registered with the JCRE, and that the client has already obtained the server delegate, as shown in Figure 6, and the client is ready to use a service from the server. The resulting client/server communication using a shared secret is illustrated in Figure 7.

The client begins by requesting a service from the server. This is done by invoking a method on the server delegate object ①. The method in the server delegate determines what level of protection is required; in this case, it determines that a shared secret must be validated to use this method. It therefore requests the delegate for the client from the JCRE ②, which the JCRE provides (if it exists) ③. Assuming the client delegate was successfully obtained, the server obtains a random chal-

lenge phrase, and sends it to the client delegate ④ using the interface the server had predefined for this purpose. The client delegate passes the challenge to the client applet (which contains the secret) ⑤. The client applet encrypts the challenge phrase using the shared secret, and returns the response phrase to the client delegate ⑥. The client delegate then passes the response phrase to the server delegate ⑦. The server delegate also encrypts the challenge phrase with the shared secret, and if the results match, the secret is validated. The server delegate then forwards the service request to the server applet ⑧, which processes the request, and returns a response to the delegate ⑨, which is forwarded to the client applet ⑩.

If the nature of the server applet is such that the services are likely to be reused by a given client in a single session, then after validation of the shared secret, the server would likely be designed to return a session key (which permits access only during the current session, which ends when power is cycled). The client could then access that particular service through the delegate by providing the session key, thus avoiding the overhead of validating the shared secret for each access of a delegate method. Although the session key is provided by the client to the server in clear text form, there is no real security risk, since the server randomizes the currently active session key each time power is cycled. So even if the session key were intercepted by an applet impersonating the server, it could not be used to breach security, due to the randomizing of the session keys.

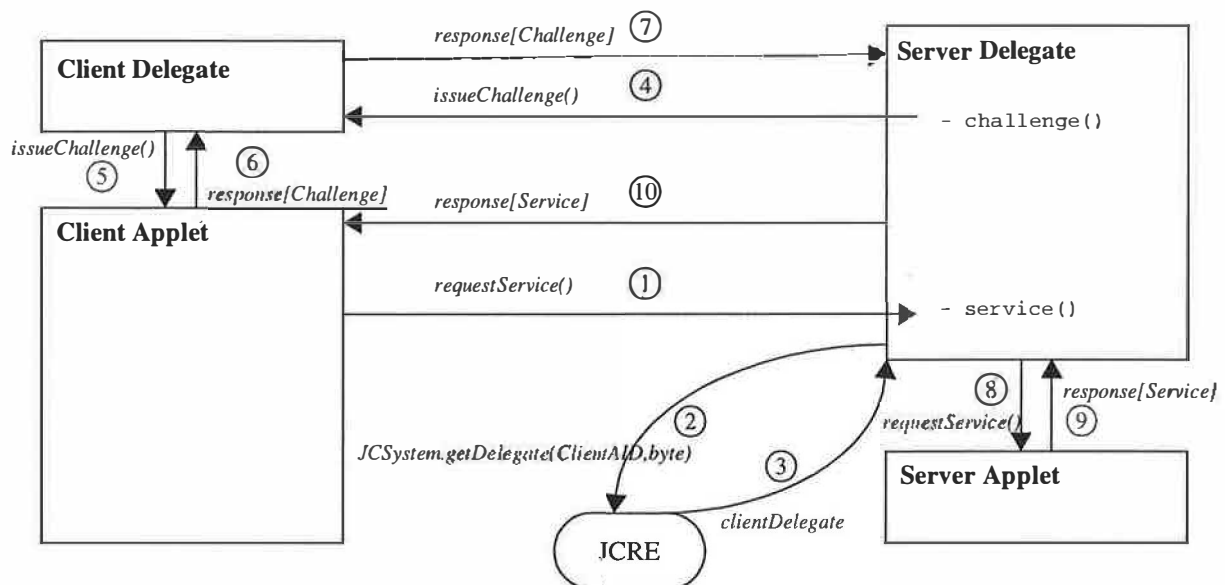


Figure 7: Challenge/Response validation of shared secret

4 Conclusion

The proposed delegate method of object sharing improves on the SIO approach. It protects each method as desired, so that no client applet gains access to unintended methods of a server applet. It avoids AID impersonation, since AIDs can be supplemented with shared secrets or other authentication mechanisms. It avoids future reference problems, since access need not be linked to any particular AID, but can simply be based on a shared secret which can easily be added to future client applications. Moreover, it requires only a minimal addition to the Java Card system. Since the delegate method allows each applet to determine the security policy desired on a per method basis, this allows the maximum flexibility for granularity of access by each individual client.

5 References

- [1] Arnold, K. and Gosling, J., *The Java Programming Language*, Addison-Wesley, 1996.
- [2] Guthery, Scott. B., *Java Card: Internet Computing On A Smart Card*, IEEE Internet Computing, pp. 57-59, Jan/Feb 1997.
- [3] Guthery, Scott B., and Jurgensen, Timothy M., *Smart Card Developer's Kit*, Macmillian Technical Publishing, 1998.
- [4] ISIO-7816, *Information Technology - Identification cards - integrated circuit cards with contacts*.
- [5] McGraw, Gary E. and Felten, Edward W., *Java Security: Hostile Applets, Holes, and Antidotes*, John Wiley and Sons, 1996.
- [6] McManis, Chuck. *My ENIGMatic Java Ring*. Javaworld, 3(8), August 1998. <http://www.javaworld.com/javaworld/jw-08-1998/jw-08-in-depth.html>
- [7] Sun Microsystems Inc., *Java Card 2.1 Virtual Machine Specification*, [//java.sun.com/products/javacard/JCVMSpec.pdf](http://java.sun.com/products/javacard/JCVMSpec.pdf)
- [8] Sun Microsystems Inc., *Java Card 2.1 Runtime Environment Specification*, [//java.sun.com/products/javacard/JCRESpec.pdf](http://java.sun.com/products/javacard/JCRESpec.pdf)
- [9] Sun Microsystems Inc., *Java Card 2.1 Application Programming Interfaces Specification*, [//java.sun.com/products/javacard/html/doc/index.html](http://java.sun.com/products/javacard/html/doc/index.html)
- [10] Sun Microsystems Inc., *The Java Card 2.1 Platform Specifications*, [//java.sun.com/products/javacard/](http://java.sun.com/products/javacard/)
- [11] Yellin, Frank., *Low level security in Java*, Fourth International World Wide Web Conference, Boston, MA, December 1995. <http://www.w3.org/pub/Conferences/WWW4/Papers/197/40.html>

Object Lifetimes in Java Card

Marcus Oestreicher
Zurich Research Laboratory
IBM Research Division
Rueschlikon, Switzerland
oes@zurich.ibm.com

Ksheerabdh Krishna
Austin Product Center
Schlumberger
Austin, TX 78726
kkrishna@slb.com

Abstract

Java Card promises the ease of programming in Java to the world of smart cards. Java's memory model however is resource intensive especially for smart card hardware. Hence, adapting Java's memory model to Java Card must retain the easy programming paradigm while enabling Java Card applications to maximize the use of smart card memory. To this end, the Java Card 2.1 Specification [3] advocates an ad hoc persistent memory model that foists an unnatural programming paradigm and an inherently limited API. In this paper we discuss memory model choices for Java Card in the context of persistent systems. We propose the concept of a transient and persistent environment for encapsulating the transient and persistent objects in Java Card applications. While offering a simple programming model, it allows efficient sharing of the memory resources among multiple applications and enables garbage collection for Java Card.

1 Introduction

The Java [1] environment possesses a number of features that make its adoption to smart cards attractive. A reasonable subset of the Java environment can constitute the base for a multifunction smart card platform. Applications can rely on standardized APIs and may be compiled into an intermediate bytecode representation enabling execution by a Java virtual machine. The simplicity and wide acceptance of the Java programming language is attractive to the smart card community where no standard language suitable for developing multifunction smart card applications has yet been established. Finally, they can benefit from the platform independence and security features provided by the Java environment.

However, Java's platform independence places significant constraints on the specific target platform. Java provides automatic memory management and does not foresee any means of manual memory control. In par-

ticular, it does not provide explicit support for persistent objects. In contrast, a smart card application requires random access to both transient and persistent memory. The limited amount of the memory resources available and their physical characteristics require simple and transparent manipulation of objects in both types of memory by the application. Hence, one of the challenges in the design of the Java Card is adapting Java's memory model to the constraints of smart card hardware.

Systems that require an intimate composition of long-lived data and programs are called *persistent application systems* [6]. Since applications in a Java Card are coupled with data, it enables the card to be a persistent application system. Orthogonal persistent systems [6] specially provide an appealing programming model for application development. Current smart card hardware typically offers around 16K bytes of persistent memory and nearly 1K bytes of transient memory. Given such constraints a Java Card cannot provide the same degree of transparency and orthogonality as an orthogonal persistent system. While it may not be desirable to completely hide the lifetimes of objects in the Java Card, it could be done in tandem with handing some control to the programmer. The necessary restrictions must be carefully chosen to provide a high degree of programmer convenience while enabling a reasonable utilization of the resources available in a smart card.

In this paper, we present the choices underlying different Java Card memory models. In particular, the differences, advantages, and drawbacks of each are highlighted. The notion of the transient and persistent environment is introduced as a solution toward simplifying Java programming on smart cards, data sharing, and support for efficient garbage collection.

The paper is organized as follows. Section 2 describes the typical memory layout in smart cards and introduces the basic execution model of applications in a Java Card. Section 3 lists the different lifetimes of objects which result from the given execution model and presents plausible allocation and placement strategies.

Section 4 discusses concepts underlying persistent systems in general and their applicability to smart cards. Section 5 outlines the approaches to introduce transience and persistence in the Java Card which were discussed during the Java Card specification process. Section 6 introduces the basic concepts and usage of transient and persistent environments. Section 7 discusses the implications of the transient environment on security, memory reclaiming and object sharing contrasting it with the approach taken by the Java Card 2.1 Specification. Finally, we present our conclusions in Section 8.

2 Smart Card Memory and Java Card Basics

Current and upcoming smart card hardware provide very limited storage capabilities. The memory resources typically consist of Read Only Memory (ROM), Random Access Memory (RAM) and Electrically Erasable Programmable Read Only Memory (EEPROM). EEPROM is used to store long-lived data. In contrast, RAM loses its contents after a power loss and is thus only available for temporary storage. Both EEPROM and RAM can be read and written; however, write operations to EEPROM are typically thirty times slower than to RAM and the possible number of EEPROM writes over the lifetime of a card is physically limited. Another difference lies in the physical size of each of these memory types. The physical size constraints on a smart card dictate RAM/EEPROM ratios often resulting in considerably smaller RAM in comparison to EEPROM.

A typical Java Card design places the operating system, the virtual machine and one or more applications in ROM. EEPROM is used to store applications which have been loaded after a card has been issued. Java Card applications, also referred to as *applets*, correspond to Java packages and are not loaded as regular Java class files [2]. Instead, a converter coalesces all the class files that comprise the package into a compact representation with minimal symbolic information. The converted code is linked on the card against the system classes and other required packages. It is up to the converter and the virtual machine to assure the Java language protection rules for downloaded code.

Once downloaded the applet is installed in a separate step. The virtual machine calls the mandatory *install* method which allocates required resources and registers a persistent object, the applet instance, with the Java Card runtime environment for future invocations. Applet execution is tailored around the server centric nature of a smart card and takes place during sessions. When a card is placed in a card acceptance device (CAD) the runtime is initialized and awaits input. Communication is handled by the underlying operating sys-

tem. An external application (the client) initiates a session with a specific applet (the server) by sending a *select* command to it. The runtime marks the selected applet active and forwards the command by invoking the applet's *select* method. Each command sent by the client hence is forwarded and handled by invoking the applet's *process* method. The applet processes the command and prepares a response which is sent after the applet has returned from its invocation. A session ends when a new select command is received. The runtime deselects the currently selected applet by invoking its *deselect* method and initiates a session with the newly selected applet.

During a session, an applet can access both the services of linked packages and services exported by other applets. A client applet can ask a server applet for a service by obtaining a reference to a shared object and invoke it freely during the session. When invoked, the server object can check the identity of the caller and grant the requested service. Note that the Java stack is completely unwound upon cessation of communication with the CAD.

3 Object Lifetimes

The object allocation, placement and invocation model influences the lifetime of objects in the system. Object lifetimes in persistent systems fall into one of the following general categories [4]:

1. *Transient (temporary) results in expression evaluations and local variables in procedure activation.*
Data in this category resides in the individual bytecode frames and is of a primitive type. Java and Java Card do not allow the explicit allocation of objects on the stack which is especially limited on the Java Card. The stack contents must only be valid during applet invocation in a session.
2. *Instance variables, class variables and heap items whose extent is different from their scope.*
Among these items are especially objects which must be accessible during applet invocation or the entire session. They must not be saved in case of power loss. For example, objects of this type are used to store the state of the communication, session keys or the communication buffer.
3. *Data that exists between two program executions.*
Objects in this category cover data which must be stored in EEPROM to survive a power loss.
4. *Data that exists between different versions of a program or data that outlives the program.*
The Java Card environment currently does not address this category.

The object lifetime categories and the memory types in smart cards give rise to the three following allocation strategies for objects in a Java Card:

1. *Objects are instantiated in RAM and are serialized by the applet into EEPROM via a file system etc.*
This strategy resembles the regular Java environment and of early Java Cards [11]. The applet instantiates objects in RAM and stores their data with the help of specific API functions into the long-term store. Each applet has to implement the functionality required for serializing and deserializing its state. There are two drawbacks with this approach. First, the working set of an applet could be larger than the available RAM. Second, the underlying operating system must manage the contents of the long-term store, i.e., it must provide a name binding with the serialized state and a means of checking the access rights of applets.
2. *Object instantiation is always in EEPROM.*
If objects are instantiated in EEPROM the language protection rules can be used to verify the integrity of the system. This provides an uniform access to all the objects of an applet. This strategy is attractive since most data manipulations are performed on long-lived data. However performing all allocations in EEPROM may never be feasible since writes to EEPROM are extremely expensive and it's life limited.
3. *Object instantiation is in RAM and EEPROM.*
The instantiation of objects with limited lifetime in RAM saves space in EEPROM, increases the performance and adds additional security in case of sensitive objects like session keys which must get lost in case of power loss. Long-lived data is placed appropriately in EEPROM. The utilization of both stores for object allocation and manipulation should still aim at the benefits of object instantiations only in EEPROM, i.e., allowing uniform access to objects and relying on Java's language protection rules.

4 Persistent Systems

The Java environment is designed as a transient programming environment. The Java Card environment however must support access to both transient and persistent objects within Java language expressions. Persistent programming languages and environments strive to enable such manipulations *transparently* [5]. Language expressions manipulating persistent data are made to appear similar to expressions operating on data with shorter lifetimes. Other than transparency, persistent

systems provide different degrees of data type *orthogonality*. Full orthogonality expresses the demand that any instance can be persistent regardless of its type and that its lifetime may not be expressed at instantiation time [6]. In any case, the lifetime of data must be easily expressible by the programmer and persistent data must be *identifiable* as such by a simple and consistent mechanism. The principles of transparency, orthogonality and identification serve as the basis for persistent systems.

Orthogonal persistent systems offer the highest degree of transparency and orthogonality and were hence chosen as the basic architecture for adding persistence to Java in the PJama project [7]. PJama allows any instance to be persistent regardless of its type and any object is identified as being persistent by verifying reachability from a persistent root set. Persistent objects are always manipulated in RAM and are lazily stabilized into the long-term store.

5 Proposed Approaches

In a Java Card objects allocated in RAM must be immediately copied into EEPROM when assigned to a persistent reference. Otherwise, unexpected power losses will lead to illegal references and loss of data integrity. Since the working set of the applet can exceed the available RAM it can only be used as a cache. However, the extremely limited resources on a smart card make it impossible to determine a suitable caching scheme which delivers sufficient results without assistance from the programmer. It is worth noting that even large orthogonal systems tend to be inefficient [10].

To counter inefficiency and provide control to the programmer, some persistent systems limit the extent of one or more principles of orthogonal persistent systems, i.e., they may restrict transparency or orthogonality [5]. The programming style is affected as little as possible. This is crucial for the Java Card which touts the simple and popular programming style of Java. Hence changes to the language to support persistence are prohibited. Introducing lifetime aware bytecode instructions to the virtual machine must be avoided as they would hinder upgrading to another memory model.

5.1 Transient Types

A common approach to introducing persistency in statically typed languages is to make persistency dependent on type. Persistent objects must be instances of classes inheriting from a specific superclass or must implement a specific interface. One proposal advocated a *Persistence* interface causing implementing class instances to be allocated in EEPROM, all other class instances would reside in RAM. Alternatively, another

approach proposed a *Local* interface to mark class instances to be allocated in RAM, with unmarked instances allocated in EEPROM.

There are two problems with this approach. Firstly, references to transient objects in the persistent set lead to dangling pointers in case of sudden power losses. Since an applet acquires access to its state by its persistent instance at invocation time, it is required to store a transient reference in its persistent set as soon as it uses a transient object. Resetting dangling pointers at the beginning of a card session involves complex scanning and is time consuming due the writes required to EEPROM. Secondly, it requires having two type hierarchies for classes whose instances may be transient or persistent. This especially restricts the use of array objects which can either be persistent or transient but never both. Additional classes simulating the behavior of fixed built-in types maybe required as well. The resulting code bloat and the performance penalty incurred by wrapper classes makes this approach unacceptable for smart cards.

5.2 Transient Fields

The introduction of separate type hierarchies may be avoided by using or extending particular language features [8][9]. Changes to the language are forbidden in Java as it affects programming style, requires educating programmers and forces changes to the Java compiler. Java however provides a *transient* keyword, a field modifier that affects object serialization. Fields marked transient are not part of the persistent state of the encapsulating object and are not serialized. It seems natural to reuse the transient modifier in Java Card to mark fields whose data must reside in transient memory and must never be saved in the persistent image of the object. The advantage of this approach is that the persistent set is only connected with the transient object set at the location of the transient fields. This allows for simpler and efficient implementations for resetting the persistent set.

The main drawback of using transient fields is it's inability to express transience in a consistent manner. The value of a reference type transient field is the reference itself. Since the transient keyword does not indicate the lifetime of the referenced object it's meaning must be extended to include the transience of the referenced object. The extended definition fails to specify the lifetime of an object which is referenced by a transient and a persistent field as well. Changes to Java semantics also prevents future introduction of the transient keyword with uniform semantics in Java Card.

5.3 Transient Data (Java Card 2.1 Specification)

Since the main requirement is to disable storing sensitive and data requiring fast access to long-term store a memory model may allocate data associated with certain objects in short-term store. The current Java Card 2.1 Specification follows this approach and is based on two basic design decisions. Firstly, applets must be designed to not expect any form of memory reclamation on the card. Applets are required to instantiate needed data at installation time and reuse it throughout their lifetime. Unreferenced data cannot be expected to be reclaimed and therefore new allocations may fail. The second design decision is to avoid dangling pointers in case of sudden power loss by expecting all objects to be referenced persistently via EEPROM. Only the data of arrays of primitive types can be allocated in RAM. As the *new* bytecodes allocate objects in the persistent store special static factory methods *makeTransientBooleanArray*, *makeTransientByteArray*, and *makeTransientShortArray* are used for allocating only the object header in EEPROM and the data in RAM.

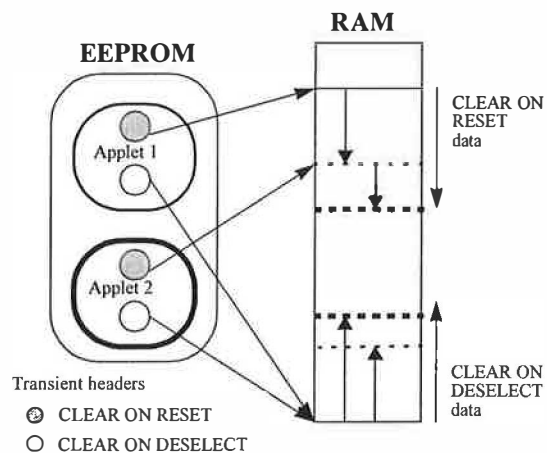


Fig. 1: RAM usage between Applets

Although the data part of such objects in transient memory is lost at power loss, the location and size information in the persistent header reserves its space over multiple card sessions. However, as RAM is shared by all available applets on the card it must be possible to limit the reservation of the transient data to less than a whole card session. Otherwise the entire RAM may be reserved after the installation of a few applets allocating transient data. The factory methods therefore take an additional argument, the duration identifier which either allows instantiations of arrays with "clear on reset" or "clear on deselect" duration. Arrays with the "clear on reset" duration will keep their values during the whole

card session, i.e., their contents are not reset between multiple applet selections during a card session. The value of arrays with the “clear on deselect” duration are always cleared before their owning applet is selected. This coarse grained lifetime specification allows overlapping the “clear on deselect” RAM space across different applets. Figure 1 shows an organization of the transient memory using this overlap. RAM is split into two segments, the “clear on reset” and “clear on deselect” spaces which grow in opposite directions. Each newly allocated “clear on reset” array gets allocated from the globally reserved “clear on reset” space while each newly allocated “clear on deselect” array gets allocated at the beginning of the per applet “clear on deselect” space. The runtime must ensure that these two spaces do not overlap.

A consequence of this design is that the entire “clear on deselect” space must be cleared at once before a new applet is selected. If a package contains more than one applet the “clear on deselect” space has to be shared by all applets as they are allowed to share their transient arrays as per the Java Card 2.1 Specification. On the other hand, an applet shall allocate all its needed data, especially its worst case usage of transient data, at install time. This can easily lead to the situation where the first applet in a package, having access to a sufficient amount of memory, will install successfully whereas the installation of a cooperating applet in the same package may fail. The problem is compounded as the runtime is unable to verify the usage of two very important memory resources pertaining to the applet, the required stack space and the worst case usage of the transaction buffer. These resources cannot be pre-calculated as they depend on the card specific bytecode frames, implementation of the required packages, and the implementation of the transaction mechanism in case of the transaction buffer. This leads to a programming model where the programmer is forced to allocate some memory resources in advance but still needs to check for their unavailability at runtime.

The lack of transient objects and the restriction on transient arrays especially effects the convenient Java programming model. A programmer is forced to load and store values from persistent objects into transient arrays and vice versa. Additionally, since resources are limited and differ from card to card, programmers have to code to a least common dominator and cannot rely on the flexible use of transient data. Not only will this force programmers to use entirely persistent objects, the larger RAM capacities of upcoming smart card hardware will ironically remain unused. The different handling of transient and persistent data also causes different transient and persistent object layout due to the indirect access to transient data through a persistent header.

6 Transient Environment

Monk: Why would anyone hurry to create gardens and buildings and monuments?

Lord Buddha: Everything is transient and nothing endures.

The problem of persistent and transient objects in Java Card can be stated as follows: *How can the persistent object model of Java Card be augmented to enable flexible use of transient objects?* The solution surprisingly lies in introducing a mechanism that is symmetrical to the persistent environment. The persistent environment consists of a persistent root, the applet instance and a set of objects reachable therefrom allocated in EEPROM. Analogously, the transient environment is formed by a tree of objects with the difference that all objects reside in RAM.

Both environments are separated in as far as references to transient objects are **forbidden** to be held in the persistent set. Storing references to transient objects in the persistent store is prevented by the virtual machine (VM) which throws an exception when such assignments are attempted. The Java (Card) bytecode instruction set uses different instructions to store primitive and reference types; only the latter must be checked by the VM, making the overhead negligible compared to the necessary EEPROM write operation in case of a store. Assignments of persistent references in transient stores need not be checked as it does not affect the integrity of the persistent store. Dangling pointers in case of power losses are avoided as transient references are never stored in the persistent set.

```
public class DummyApplet extends Applet {
    public boolean select() {
        JCSystem.beginTransience();
        Object o = new Object();
        JCSystem.endTransience();
        JCSystem.setTransientEnvironment(o);
        return true;
    }

    public void short process(APDU apdu) {
        Object o;
        o = JCSystem.getTransientEnvironment();
        // use o
    }
}
```

Fig. 2: Using Transient Environments

The lifetime of an object is controlled with the help of an API mechanism that demarcates transient allocation defaulting to persistent allocation when not called. As shown in Figure 2 the API is used to toggle the allocation mode. Allocations between *beginTransience* and *endTransience* are in RAM, defaulting to EEPROM.

RAM overflow is signaled by an exception similar to how EEPROM overflow is signalled. This mechanism resembles a typical transaction API wherein behavior of state manipulation is toggled between a *beginTransaction* and *commitTransaction* code section.

Allocated transient objects may be interconnected to form a transient environment whose root can be registered with the Java Card runtime by invoking the *setTransientEnvironment* method. Where a particular persistent environment is implicitly made available to the applet during invocation of the process method, the applet can request its transient environment by explicitly invoking the *getTransientEnvironment* method. This allows accessing transient objects even when assignments of transient object references to persistent fields are prohibited.

The transient environment naturally fits in the current execution model of Java Card applets. It may typically be built in the beginning of a session in the select method. Transient data which must survive the session is copied to the persistent store at the end of the session. After the session, the transient environment is reset and the designated RAM space is available for the next applet session. The ease of creating transient objects of any type leads to a convenient programming style, results in better performance and more compact code. The similarities between the transient and persistent environment also simplifies virtual machine implementations where the same object layout may be used for both the transient and persistent objects.

7 Implications

Apart from enabling a convenient programming mechanism the transient environment scales into the future when more smart card resources become available. The only limitation introduced by the transient environment is the restriction on assignment. Future systems may choose to remove this restriction and still provide binary compatibility. The restricted assignment also enhances security in that an applet cannot store a reference to a transient object which it received from the system or from a different applet in its persistent set. Since the object is only accessible during a session it cannot be accessed in situations not foreseen by the service provider. For instance, it allows the deletion of a server applet without the danger of dangling pointers in the client applet. The transient data approach is not upgradable in large part due to the fixed lifetimes in the API. The transient space is statically split for the individual applets which make temporary allocations and deallocation in the future practically impossible. Lifetimes contradict the ease-of-use of standard Java.

The transient environment also strengthens other aspects of the Java Card runtime, especially memory reclamation and the sharing mechanism.

7.1 Memory Reclamation

While not addressed by the Java Card 2.1 Specification memory reclamation can be very effective for the smart card environment. Clearly manual memory reclamation cannot be allowed due to the security implications and must be avoided in favor of garbage collection. Surprisingly, EEPROM write performance and not the size of a general garbage collector is a hindering factor with regards to a memory reclamation scheme. For instance, a simple mark and sweep garbage collector first annotates all reached objects and frees all unreferenced objects in a second pass [12]. It is most likely due to the limited RAM size that the annotation information, i.e. the mark bits, must be written into EEPROM. The resulting performance penalties make it impossible to interrupt the applet execution for garbage collection as soon as memory is scarce but only at fixed points in time. However, cleaning up EEPROM is not as critical as that of RAM.

7.1.1 Transient Environment Garbage Collection

Typical applets allocate their persistent set at installation time and limit the changes therein to data update. New instantiations or complete replacement may be considered rare. The RAM space is limited and can be consumed quickly as soon as an applet allocates transient objects which are not reused in the transient environment but allocated only for the duration of an applet invocation. However, in case of transient environments garbage collection is not only feasible but indeed practical. The transient environment can be garbage collected completely separately from the persistent environment. Persistent objects need not be scanned as their fields never reference transient objects. Our implementation achieves satisfying results when invoking the garbage collector upon return from the applet's select, deselect, or process methods. The Java stack is empty and the root set for the garbage collection consists only of the transient environment.

The same garbage collector may be used for cleaning up the EEPROM. Due to the limited performance this should only be carried out after an explicit request by an applet or by a special command from an external application.

7.1.2 Limitations of Transient Data Memory Reclamation

EEPROM garbage collection is also permitted by the transient data approach. However, a different mechanism must be used for reclaiming RAM space. The runtime can attempt to reuse the space for globally allocated “clear on reset” arrays after their applets have been deleted. It may also allow the increase of “clear on reset” space after the applet with the most “clear on de-select” usage has been deleted. The added complexity stands in contrast to the low amount of memory which can be reclaimed generally in this static environment.

7.2 Sharing

The flexibility of the transient environment is not only afforded by the selected applet but also by the services it uses. The Java Card environment distinguishes and supports three different sharing scenarios:

1. *An applet is linked against a separate package.*
The package contains shared code used by different applets to create instances of classes and invoke methods in this package.
2. *An applet has a reference to a shared object provided by another applet.*
The client applet requests the reference from the runtime which forwards the request to the serving applet and returns the received reference to the client applet. The reference must be an interface type and the virtual machine will refuse any other attempts to access the methods specified by the interface.
3. *An external application collaborates with two or more applets on the card.*
The applets know about the collaboration and want to keep access to their transient state as long as the collaboration lasts.

The transient environment can provide a flexible use of transient data in all three scenarios. The degree of flexibility depends on the runtime providing support for only one transient environment, multiple transient environments and/or garbage collection.

7.2.1 Single Transient Environment

The transient environment plays well in the first scenario wherein a shared package can either be given access to the transient environment of its client applet or can build its own transient environment. In the first case, the applet and package transient environment must obey Java type rules which involves the applet subclassing its environment root class to the package environment root

class. In the second case, where the transient environment of the applet and the package differ, the applet can use a simple mechanism to adopt its transient environment to contain the package’s transient environment. The applet has to reserve one node in its environment for the root of the package environment. The first time it calls into the shared package it saves its current environment in a local variable and resets its environment at the runtime. Within the call the shared package can create an environment for itself, register it and fulfill the requested service. Upon return the applet can save the package environment in its designated node and register its original environment. From now on the applet must always save its environment on the stack and register the package environment before it invokes the target package. If the runtime provides a garbage collector, both the applet and the shared package can allocate local objects which are not part of the environment and are garbage collected after the current invocation of the applet. As the applet is in control of the shared package environments, it can reset their roots any time during the session and thus subject them to garbage collection. This allows the optimization of memory usage for instance when an applet uses multiple packages alternately during a session. The same mechanisms apply in the second scenario.

The third scenario demands extending the transient environment lifetime over an applet session. This is achieved by either requiring the applet to not reset its transient environment at the end of the session or using a system method to retrieve it. The longer living environment can then be shared by the collaborating applets. They manipulate it alternately until the last deselected applet during the collaboration finally resets it. The runtime may then free the transient space for the next session.

7.2.2 Multiple Transient Environments

Collaborating applets may have different transient environments causing the runtime to support multiple transient environments at a time. Each collaborating applet creates and registers its own transient environment with the runtime system and extends its lifetime to last longer than its current session. The runtime switches between the transient environments whenever an applet is deselected and the next applet during the collaboration is selected. When the last deselected applet resets its transient environment the runtime releases the transient space. If a garbage collector exists, parts of the transient space can be reclaimed when any applet resets its transient environment.

The support of multiple environments can be useful in the second scenario to simplify programming and to

encourage the full use of all available RAM. When a client applet requests a reference from a server applet, the request is forwarded by the runtime to the serving applet. The server applet creates and initializes its own transient environment, registers it with the runtime and returns a reference to the requested shared reference. The runtime marks the serving applet as being part of the current session and forwards the reference to the client applet. Whenever the client invokes a method on the server reference the runtime switches to the appropriate transient environment. The server object can access its transient environment and use it to execute the requested service. When the client applet is finally deselected, all transient environments of the participating server applets are also reset.

As opposed to the single transient environment the management of the individual environments is up to the Java Card runtime. This simplifies the programming model but limits the control of the executing applet over memory utilization. The Java Card runtime is not able to release any of the participating environments prior to the end of the session without the support of the shared services or the request of the currently selected applet.

7.2.3 Limitations of Transient Data Sharing

The static memory model of the transient data approach fails to provide a flexible use of transient data in any of the three scenarios.

For the first scenario, a shared package can only allocate transient data if it is called during the installation of the applet. Moreover, it must connect its transient data to the persistent set for later use.

For the second and third scenarios, both the shared object and the collaborating applets have to keep their transient information in globally reserved “clear on reset” arrays. A shared object cannot use the “clear on deselect” arrays of its implementing applet as the “clear on deselect” space is reserved for the currently selected applet, the client applet. This forces either the global reservation of RAM by allocating “clear on reset” arrays or the renunciation of transient data. In the first case shared applets can hog RAM causing denial of service problems by preventing installation of any client applet. In the second case the performance of EEPROM will prevent the sharing of any complex services and force each client to implement parts of or even the whole service, defeating code reuse.

8 Conclusions and Future Work

The contributions of this paper are:

- We suggest a terminology and framework with which to describe the issues underlying memory models in Java Card.
- We identify the restriction that can be placed on orthogonal persistent systems while retaining all the benefits of persistent systems for the Java Card programmer.
- We have presented other solutions attempting to solve the problems of transient data and shown their weaknesses. In particular we have shown that the static memory model described in the Java Card 2.1 Specification results in an unusual programming model and restricts the possibilities for memory reclamation and object sharing to an unsatisfying degree.
- We have shown that, by making support for transience explicit, a Java Card can provide a scalable API that can allow the manipulation of transient data similar to the standard Java environment. The resulting dynamic memory model naturally fits Java Card’s execution model, allowing the simple and effective deployment of a garbage collector and enhances object sharing.

The proposed environment has been implemented and tested on a number of applets. In the future we hope to provide concrete benchmarks supporting its simplicity and performance to enable comparison of the design choices. We hope the transient environment will further demystify smart card programming and permit Java Card programmers to truly enjoy the benefits of persistence.

9 Acknowledgments

We are deeply indebted to Peter Buhler and Michael Baentsch for their invaluable input on implementation and many productive discussions. We are grateful to Stephan Hild for carefully reviewing the paper.

10 References

- [1] Arnold, K. and Gosling, J., *The Java Programming Language*, Addison-Wesley, 1996.
- [2] Lindholm, T. and Yellin, F., *The Java Virtual Machine Specification*, Addison-Wesley, 1996.
- [3] Sun Microsystems Inc., *Java Card API 2.1 Specification*, [//java.sun.com/products/javacard/JavaCard21API.pdf](http://java.sun.com/products/javacard/JavaCard21API.pdf)

- [4] Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. and Morrison, R., *An approach to Persistent Programming*, Computer Journal, 26(4), 360-365, Nov. 1983.
- [5] Hosking, A. L. & Moss, J.E.B., *Approaches to Adding Persistence to Java*, Proceedings of the First International Workshop on Persistence and Java, Drymen, Scotland, Sept. 1996.
- [6] Atkinson, M.P. and Morrison, R., *Orthogonally Persistent Object Systems*, VLDB Journal, 4(3), 1995.
- [7] Atkinson, M.P., Daynès, L., Jordan, M.J., Printezis, T. and Spence, S., *An Orthogonally Persistent Java*, ACM SIGMOD Record, Dec. 1996.
- [8] Hosking, Anthony L. and Moss, J. Elliot B., *Compiler Support for Persistence*, COINS Technical Report 91-25, March 1991.
- [9] Schuh, Dan, Cory, Michael and Dewitt, David, *Persistence in E revisited - Implementation Experiences*, Proceedings of the Persistent Object Systems Workshop, Martha's Vineyard, MA, September 1990.
- [10] Cooper, Tim and Wise, Michael, *Critique of Orthogonal Persistence*, International Workshop on Object Orientation in Operating Systems, October 1996.
- [11] Guthery, Scott. B., *Java Card: Internet Computing On A Smart Card*, IEEE Internet Computing, pp. 57-59, Jan/Feb 1997.
- [12] Jones. R. and Lins. R., *Garbage Collection, Algorithms for Automatic Dynamic Memory Management*, Wiley, 1996.

A Personal Naming and Directory Service for Mobile Internet Users

Alain Macaire & David Carlier
Gemplus Research Lab.
BP 100 - 13881 Gemenos - FRANCE
cameleon@research.gemplus.com

Abstract

This paper proposes a new approach for the role of smartcards into distributed and mobile service environments. It is based on the naming and directory service architecture. We present a naming and directory service architecture which is based on a new component we named *Personal Naming and Directory Service* (PNDS), which is embedded on a smartcard. In section two, after a short introduction, we present PNDS concept and list advantages to have it stored on a smartcard. Section three gives an overview and limits of current smartcards applications for mobile users. Section four presents PNDS features more precisely, and shows how it has been integrated into a federated architecture of naming servers (PNDS has been prototyped using a GemXpresso JavaCard platform). To demonstrate the PNDS concept, an example of a PNDS-based application is presented in section five.

1 Internet Services and User Mobility

With the growth and spread of the Internet, vast information resources and services are making available to anyone, at any time, from anywhere in the world. People and businesses are becoming increasingly dependent on rapid and easy access to information drawn from both local and global sources. As more and more people and places become "connected", technological needs and market

forces continue to change at an increasing pace.

The Internet community currently focuses part of its forces on the convergence of fixed and mobile networks to provide access to the Internet from wireless terminals (e.g., cellular phones, pagers, in-car computers, palm-top computers). Internet Engineering Task Force (IETF) is chartered to develop or adopt architectures and protocols to support mobility within the Internet [1]; World Wide Web Consortium (W3C) is working towards making information on the World Wide Web accessible to mobile devices [2]; WAP Forum (Wireless Application Protocols) [3] is defining de-facto world standard for wireless information and telephony services on digital mobile phones and other wireless terminals; Co-operation between W3C and WAP Forum has started around a common test bed [4].

This convergence of system and network infrastructures leads to offer on-line access to mobile users regardless their physical location and the serving network, and through various types of terminal devices having different capabilities and interfaces. Therefore, mobile users will need intelligent information handling to easily access information and services and customize terminals and applications according to their own preferences (user profiles).

This paper proposes a new approach for the role of smartcards into these distributed and mobile service environments. This approach is based on the naming and directory service architecture. We present a naming and directory service architecture which is based on a new component we named *Personal Naming*

and *Directory Service* (PNDS), which is embedded on a smartcard. In section two, after a short introduction, we present PNDS concept and list advantages to have it stored on a smartcard. Section three gives an overview and limits of current smartcards applications for mobile users. Section four presents PNDS features more precisely, and shows how it has been integrated into a federated architecture of naming servers (PNDS has been prototyped using a GemXpresso JavaCard platform). To demonstrate the PNDS concept, an example of a PNDS-based application is presented in section five. Finally, we conclude with future directions.

2 Naming Services

With the emergence of the Internet and distributed object technologies, naming services have become essential elements in distributed system architectures. *Naming Services* make possible communication, data exchange and co-operation among different distributed objects by providing name-to-object resolution. Moreover, naming services provide foundation for more evolved services such as *Directory Services* and *Trading Services*.

2.1 Naming & Directory Services

Naming services provide objects from a request which has a name as an argument. A naming server manages a hierarchical structure of objects, and provide navigation facilities over a logical graph naming of contexts (figure 1).

In addition, a directory server manages a collection of attributes for each registered object. Attributes hold characteristics of objects and allow servers to provide client with powerful search and filtering mechanisms on attributes, hence on objects. Clients specify search criteria with their requests, and get in return a list of objects which match those criteria.

Naming and directory services can be

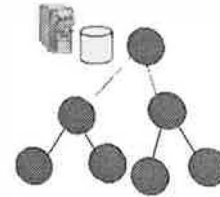


Figure 1: Graph of Naming Contexts

viewed as special address books which are distributed across the network and which provide information on distributed objects. Objects may be of different types such as for example IP addresses from the Domain Name Service (DNS) [5], CORBA Interoperable Object References (IOR) [6], corporate directory entries from an LDAP database (Lightweight Directory Access Protocol) [7], or personal directory entries from a personal address book.

The list of attributes along with their types depend on the type of registered objects. For example, in case of an address book, **e-mail** and **phone-number** are attributes of a **person** entry; in case of a network-printer directory service, **printing-quality** (laser vs. dot-matrix) is an attribute for a **printer** entry; in case of a user profile, **preferred-colors** and **languages** are attributes for a particular user service.

Combining objects with attributes allows servers to provide each time an adapted service. *Trading Services* benefit from these features and provide users with features to discover and access to new services according to their types and characteristics.

2.2 LDAP

Even if many naming servers have already been implemented for a while such as Domain Naming Service (DNS), Network Information System (NIS), or CORBA naming service (COS), LDAP is a new emerging naming and directory service.

The main interest of LDAP consists of flexibility. All current naming services can be implemented with this protocol. The structure of an LDAP service is based on a hierarchy

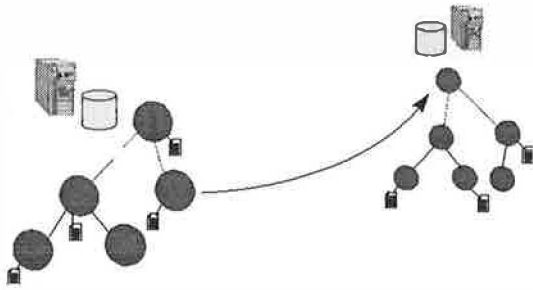


Figure 2: Referral vs. Referred Context

of entries made of attributes and bound objects. The lookup of objects and the search according to filters on attributes provide with convenient accesses [8]. Access controls are supplied by identification and authentication.

Interesting features of LDAP include the support of *Referrals Contexts*. This special type of entry is used to forward requests to other naming servers on the network when the current server cannot provide with the requested object. With referrals, different naming spaces from different naming servers can be linked together (figure 2). Referral entries also allow to share data among several users and make easier global updates on distributed databases.

We have chosen LDAP protocol as a reference for the *Personal Naming and Directory Service* (PNDS).

3 PNDS

Naming and directory services are traditionally supported by network servers and are provided to users as part of their network and service provider subscription.

However, on-line connections and services evolve to become more personalized to users and available at anytime from anywhere. The concept of *Personal Naming and Directory Service* (PNDS) was developed to provide mobile users with the part of naming and directory service that may be private and personalized. PNDS is implemented on a smartcard and is fully integrated in the overall

naming and directory architecture through referrals (figure 3).

PNDS is a generic component which is able to store a hierarchical directory of bound objects along with pairs of attribute-value. Therefore, PNDS is perfectly suited to store various kind of users' or network related data, such as for example :

- object references (e.g., network addresses) which allow the system and network to bind to remote services,
- user service profile entries, which personalize services the user has subscribed to,
- Users' personal applications such as for example a personal address book.

3.1 Three Modes of Operation

The PNDS leverages the LDAP concept of referrals by handling three modes of operation.

1. When set in the *Referral Ignore* mode, PNDS ignores every referral, and directory lookups are performed locally in the smartcard. This is especially useful when the network is unreachable, or if the user does not want to open a network connection.
2. When set in the *Referral Throw* mode, PNDS throws an exception at destination to the client application as soon as it traverses an object bound to a referral. The client application can choose to open a network connection, and request from the PNDS the remaining part of the query to complete the lookup, as well as the address to contact the server.
3. When set in the *Referral Follow* mode, PNDS is able to follow referrals on its own. Without informing the client application that the requested object is located on a remote server, PNDS requests the hosting terminal to open a network connection and forward the request.

An example of using such a feature is when the user wishes to access a specific service. As the required service information may already be stored on the smartcard (service profile), the first lookup to the PNDS can be performed using the *Referral Ignore* mode. Depending on the result, a second attempt will be issued using *Referral Throw* or *Referral Follow* modes, to link to the network and retrieve service profile information from the referred server.

Data from the PNDS can be updated either by service providers/administrators from the network, or directly by users themselves from the client application on the terminal. A security model for access controls will have to be provided (see section 7). Therefore, it will be possible to bookmark the result of queries locally on the PNDS smartcard for next uses.

3.2 Remote Attributes

Due to their tiny size, smartcards have inherent limitations in term of memory capacity (see section 4). Thus, we have introduced the concept of *Remote Attribute* to reference object attributes which are located remotely on external content servers (figure 3). Commonly, a reference attribute will be stored as a URL, but any other addressing schemes can be supported (e.g. phone number¹).

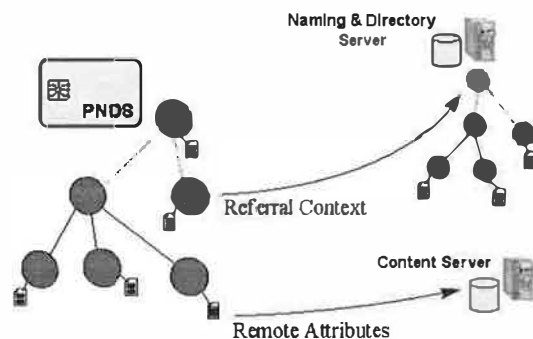


Figure 3: A Personal Naming & Directory Service

¹It is possible to use URL addressing scheme to reference any content and services, as for example in WTA telephony services specification from the WAP Forum.

4 SmartCards for Service Personalization

A smartcard is a plastic card with an embedded microprocessor and memory which allows it to store data and execute code. The main concerns about smartcard include data confidentiality, secure authentication and high computing mobility (due to its small and convenient size). The current limits are restricted memory (up to 32Kb).

Current types of smartcards are based on either a file system [9], a small SQL-based database [10], or a virtual machine based operating system such as the JavaCard [11]. This last type of smartcards allows service downloading and is well-adapted for the development and deployment of new applications. Development and integration of services in such a JavaCard can be full object-oriented, hence the integration in distributed systems is made easier. Therefore we have chosen the Gemplus GemXpresso JavaCard, which provides full object-oriented design and programming model [12].

4.1 Current Applications for Mobile Users

As far as network access is concerned, the SIM card [13] is certainly today the most widespread smartcard. SIM cards allow mobile users to access the GSM network [14]. Upon entering a PIN code, user is identified and authenticated, and access is granted whatever the GSM terminal used. Moreover, the SIM card is used to store and provide users and terminals data such as personal address books and small interactive applications [15].

More recently, smartcards which support public-key algorithms are being deployed to provide Internet security to users [16]. These type of cards generate the private and public keys on their own. The public key can then be exported as a certificate (e.g., X500), while the private key will never be released outside of the card.

4.2 Limits of Current Network-Oriented SmartCards

Current smartcard applications allow terminal personalization. When the card is inserted, an anonymous terminal can become a personalized terminal. However, this personalization capability is still not widely used. Main smartcards concern is limited to security.

Smartcards provide data storage, but currently only the file and directory structures for binary data is deployed. This reduces the role of the card as a simple binary data server and therefore such a card cannot act in a full co-operation within architectures of open terminals, networks, systems and services.

The naming approach applied to the smartcards makes them more adapted to distributed environments. A naming environment provides to the terminal all personalized information with a better interface than a file system. Powerful searches and referral entries make easier the integration of such a smartcard into distributed systems.

4.3 Benefits for Mobile Users

PNDS extends users' mobility because the part of users' personal and private information is easily and securely carried-on from terminal to terminal. The benefits for mobile users are at least threefold.

- **Access from different access points and terminals:** Users can access services from different terminals and locations (e.g., network computers), and keep each time their own personalized features. Also, services can be adapted according to resources available locally.
- **Access in stand-alone :** Users can access part of their private and personal information securely, even in stand-alone mode, without any network elements or data involved.
- **Security:** The potential threat to individual privacy makes end-users wary

about sharing personal information [17]. Storing personal and private information into a secure and tamper resistant device (i.e., a smartcard), allows the control of information exchange by means of user, application, and/or system level authentication.

5 A PNDS Implementation

The PNDS is an individual naming server with a similar approach to LDAP embedded into a smartcard². This service must be supplied in any circumstances. The smartcard is an appropriate support to provide such a personal naming server for mobile users, as it contains the personalization part of mobile users' services.

A directory structure like LDAP appears to be a solution to propose different naming spaces to different services. A naming space is defined by a directory entry.

5.1 GemXpresso JavaCard

The PNDS has been prototyped on a GemXpresso: the Gemplus' 32-bit RISC JavaCard. This card allows one to easily write a card applet in Java language and to invoke it through a generated Java proxy. The client application invokes Java object methods without being aware of the specific smartcard commands.

The PNDS is made of directory entries in a hierarchical structure. Each entry contains a list of attributes. The design was made with the concern to save memory in the card and to have a better execution speed. Thus, a special attribute is referenced as the entry name, and binding an object to an entry is performed

²Even if the PNDS is similar to a smartcard-embedded LDAP server, the set of commands is specific. Due to today's smartcard memory capacity limitation, a real LDAP implementation into a smartcard is impossible. Furthermore smart card communication protocol is different (actually it is not TCP/IP). Thus, the integration of such a server appears to be an impossible challenge.

by adding an attribute. An attribute consists of a name-value pair. For example, an entry corresponding to a person description could be as it follows.

```
cn = Durand /* common name (entry name) */
gn = Pierre /* given name */
l = Paris /* location */
pn = +33 4 12 34 56 78 /* phone number */
m = pdurand@gemplus.com /* e-mail */
```

Searching an entry in a directory is carried out by a search engine implemented in the smartcard. This search engine provides a list of entries matching attribute criteria.

A request to get, add or modify an entry is allowed only after being authenticated by the user's password.

5.2 PNDS Integration into Distributed Systems

A common interface for Java objects has been defined by several international companies to unify the access to different types of naming and directory services. This interface is named JNDI [18] (Java Naming and Directory Interface). JNDI supports are available for major naming and directory servers (e.g., X500, LDAP, NIS, COS).

We decided to choose JNDI to realize the PNDS integration into distributed systems because this model proposes a way to federate all naming servers. PNDS appears to be just a new naming and directory server to be integrated within this framework (figure 4).

In addition to the PNDS embedded-smartcard server, and like the other naming services, we have developed a JNDI SPI (Service Provider Interface) [19], outside the smartcard, to request the PNDS. A client application invokes JNDI methods to the PNDS like other naming servers without being aware of PNDS smartcard requests.

For example, a client application can call the `lookup()`, the `getAttributes()`, or the `list()` methods, to lookup, retrieve attributes, or list sub-entries of

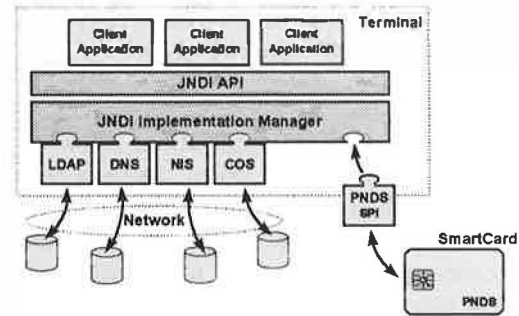


Figure 4: PNDS Integration into JNDI

a particular entry. Also, by calling the `createSubcontext()` or the `modifyAttributes()` methods, the client application can create new objects/entries in the graph, or create, modify or remove attributes of an entry.

The `DirContext` Java class, which has been implemented, converts standard JNDI commands to PNDS requests to a GemXpresso smartcard proxy. Implementation of the `InitialContextFactory` JNDI interface provides PNDS initial contexts, that is to say the way to access and send requests to the PNDS.

5.3 Federating Naming Services

As explained previously, implementing a JNDI interface to the PNDS supplies client applications with a unique interface. However the JNDI API allows the management of referrals. A JNDI referral context references another context. This context may be on the same server or on a different one. This other server may host a different type of naming service (e.g. COS, LDAP, ...).

A PNDS entry is a referral entry when it contains a referral attribute with a reserved name and an address to the referred context. When a request needs to explore an entry bound to a referral entry, the behaviour of PNDS depends on the mode of operation that is currently set³:

³A different operation mode can be set at each request.

- **REFERRAL_THROW** or **REFERRAL_FOLLOW**⁴: an exception is raised to the PNDS-SPI interface. This interface generates a JNDI `ReferralException` to the client application references and data delivered by the PNDS. The client can invoke the `getReferralContext()` of this exception to easily get the referred context.
- **REFERRAL_IGNORE**: PNDS ignores the referral and continues to descend the directory hierarchy, trying to find the requested entry locally. Depending on the result, the requested object or `NOT_FOUND` is returned.

The PNDS is not only a naming and directory server, but also an access point to other naming servers. The PNDS contains both personal named objects and links to other named objects managed by other servers outside the smartcard.

6 Example of PNDS Application

To demonstrate the PNDS features and capabilities, we have developed a personal address book as an example of an application (figure 5). This PNDS-based application contains two parts.

1. The first part, the address book itself, supplies mobile users with a set of person information, classified in a hierarchical structure composed of person entries and directory entries. Entries which are private to the user are stored locally on the PNDS smartcard, while shared or public entries are bound to referrals.
2. The other part provides user profile information dedicated to personalize the address book user interface on the terminal according to user's preferences.

⁴**REFERRAL_FOLLOW** has not yet been implemented. Smart cards such as those supporting the SIM Toolkit API [15] are appropriate candidates for implementing this type of card-driven action towards the terminal and the network.

(Note that in our example, we did not use any referral or remote attribute in user profiles, but of course this can be possible.)

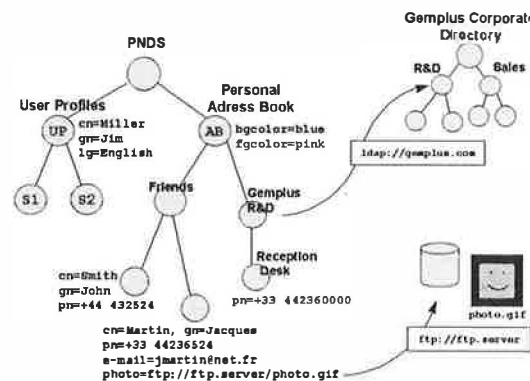


Figure 5: the Personal Address Book

This application is an appropriate PNDS demonstration. It must be available from anywhere whatever the terminal used. As opposed to the address book application code, address book data are personal. This requires features showing the advantages of PNDS and smartcards.

Via the PNDS, the address book application accesses data located on external naming server without being aware of requests to remote servers. Actually, as the address book data are distributed over the network, the PNDS opens the door to a worldwide personal address book.

6.1 PNDS as a Personalized and Private Secure Data Server

The PNDS provides information which enables users to browse their address book over a hierarchical structure of entries, from a directory entry to another. Users can select a person entry (a leaf of the structure), to retrieve its related information. A person entry consists of a set of attributes such as the common name (`cn`), given name (`gn`), phone number (`pn`), e-mail (`mail`) and photo (`jpegphoto`).

The PNDS can be viewed as an opened and

powerful secure data server, not only as a simple and closed secure data provider such as in a traditional file system. The PNDS is a full naming and directory server.

In addition to the `lookup()`, the PNDS provides a `search()` command to find out person(s) into the directory structure according to conditions on attributes. For example: `gn=Pierre` or `pn=+33 4 42*`.

Furthermore, the semantics of some entries may be different. For instance, Gemplus corporate directory has not to be stored inside the smartcard because it is provided and supported as part of the Gemplus' information system on the network. Thus in our case, just a referral entry is stored in the PNDS for Gemplus R&D directory entry. When this entry is selected, PNDS provides all necessary information to the JNDI naming manager to transparently forward requests to Gemplus' server (`REFERRAL_THROW`). If the mode is set to `REFERRAL_IGNORE`, PNDS provides the `Reception Desk` entry (`pn`).

Finally, some attributes of a person entry may be too large with respect to smartcard features. For example, attributes such as pictures or home-pages cannot be stored on the smartcard. However a `picture` attribute is bound as a remote attribute to the picture provided by a content server on the Internet.

The PNDS smartcard is one part of a personal address book distributed on the Internet. PNDS acts as a federation component.

6.2 User Profile Management

PNDS provides also an appropriate support to personalize an anonymous terminal with user profiles. We have decided to manage two levels of user profiles:

- a general user profile, which contains information related to user's general description and preferences,
- a service-specific user profile, which contains information to personalize a particular service.

In our example, the general profile is implemented by a directory entry created as a subdirectory of the PNDS root. This entry named `UserProfiles` (`UP`) provides information attributes related to the PNDS holder such as his/her common name (`cn`), given name (`gn`), and preferred language (`lg`) (figure 5). All these data describe the user.

We chose to store user profile information related to the address book directly as attributes to the `PersonalAddressBook` entry (`AB`). These attributes consist of the information to personalize the address book user interface according to the user's preferences such as foreground color (`fgcolor`), and background color (`bgcolor`) (figure 5).

Also, when services themselves are not part of the PNDS smartcard, user service profiles can be specified by creating new `UserProfile` subdirectory entries (e.g., `S1`, `S2` on figure 5).

6.3 Perspective of this Application

As an extension to our demonstration, we plan to integrate our personal address book in an e-mail manager such as Netscape Communicator which supports access to LDAP directory servers on the Internet. In the latest 4.5 version, this application supports the notion of *Roaming Access* for roaming users to retrieve user profile information from any place on the network.

At each new connection from an anonymous terminal (e.g. a public network computer), users must manually enter an LDAP server address along with a distinguished name (`dn`), in order for the application to retrieve their profile and set their preferences accordingly. However, since relatively few users know their distinguished name, and probably fewer can type it correctly, this manual configuration may be tedious and can easily be replaced by just inserting a smartcard into a smartcard reader.

Also, accessing the Internet from anywhere at anytime should allow users not only to retrieve their personal preferences but also to

benefit from features and services which are available locally, provided as part of the local network or service provider.

A plugin can allow Netscape Mail to access the address book service from the PNDS, and update the current configuration. Thus, when users insert their card into a terminal, the Netscape mail address book will be personalized from the PNDS.

7 Perspectives on Security

Security should play a central role in the Personal Naming and Directory Service. However, scope of this paper is only limited to describe PNDS itself and its integration into distributed systems. Therefore, we focus our discussion on presenting only some possible mechanisms to deploy security within the PNDS. We consider the following security concerns :

- Controlling the accesses to the PNDS and its data,
- The role of the PNDS in the overall security architecture of a distributed application.

7.1 Access Controls

Access to the PNDS information is currently permitted after typing the right PIN code, nothing is supplied otherwise. However a PNDS may consist of several services for various external applications with different types of accessing users. Access to pieces of information may require a specific authorisation.

A first approach of this problem may lead to identify two kinds of users, each one having a different level of access privileges to read/write parts of the PNDS :

1. a **cardholder level**, which allow users to modify entries from their personal

profiles and applications (e.g., Personal Address Book);

2. an **administrator level** (i.e., network and/or service providers), which allow service/network providers to remotely manage (update) service profile entries.

Different PIN codes can be assigned to different privilege levels, and access conditions have to be set at the context level.

7.2 Security Architecture

The other perspective concerns the overall security of distributed applications. Extensive security can be implemented for naming and directory services. PNDS can act as a keys and certificates provider, and is able to use cryptographic features provided as part of the smartcard operating system.

Possible roles of PNDS in the security of distributed application over the Internet are illustrated on figure 6. The *Secure Socket Layer* (SSL) is used to authenticate users to other naming servers on the network (i.e., referrals), while the *Remote Keys Encryption Protocol* (RKEP) [20] is used to secure content (i.e., cipher/decipher mail folders). Part of such a security architecture has already been demonstrated by Gemplus in the Vault prototype [21].

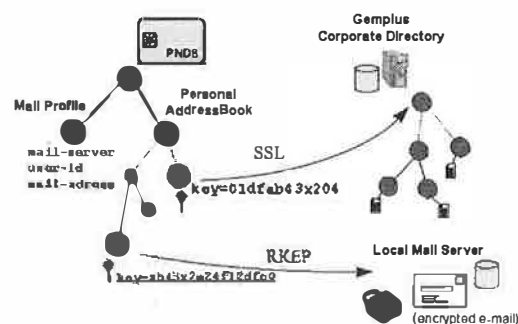


Figure 6: Example of PNDS-based Security Architecture

8 Conclusions

Providing an adapted and personalized service is not limited to just only taking into account users' preferences. With the convergence of different network infrastructures, systems and terminals, this problem encompasses network profiles, terminal profiles, and of course smartcard profiles. These include hardware and software profiles and finally users' profiles, which can be partly carried-on within the smartcard. Providing a service matching the devices capabilities and users' preferences at each connection, is one of the today's challenge on network convergence (figure 7).

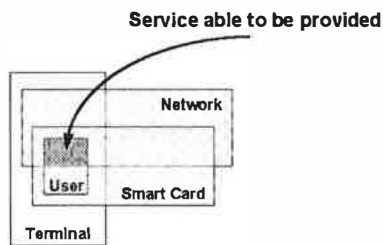


Figure 7: Adaption of Services

PNDS-based smartcard is an interesting concept for profiling aspects. It proposes a flexible structure based on a naming and directory service for applications and systems on the terminal. PNDS allows the integration of user profiles in an anonymous terminal, and personalizes the terminal and its applications with both information stored inside the card and references to personal data on the network.

PNDS is a generic component that is able to store any kind of objects, and referrals to objects accessible on the network are implemented both to get an infinite data memory capacity and to share data between several people.

Furthermore, PNDS has been integrated in a global framework and architecture based on a unified application programming interface (API). This means that client applications invoke PNDS as any other servers, without being aware of PNDS smartcard specific commands.

In this prototype, access control has been limited to PIN code authentication. Next step would be to refine the security, and define a security model for accessing and modifying PNDS local and remote objects.

9 Acknowledgements

This work was partly supported by the Commission of the European Community (CEC) as a part of the CAMELEON ACTS research project (AC341).

References

- [1] Routing Area Working Groups, *IP Routing for Wireless/Mobile Hosts (mobileip)*, Internet Engineering Task Force (IETF), <http://www.ietf.org/html.charters/mobileip-charter.html>.
- [2] Mobile Access Interest Group, *Working towards seamless Web access from mobile devices*, World Wide Web Consortium (W3C), <http://www.w3.org/Mobile/>.
- [3] Wireless Application Protocols Forum, <http://www.wapforum.org>.
- [4] J. Hjelm, B. Martin, P. King, *WAP Forum - W3C Cooperation White Paper*, W3C, <http://www.w3.org/TR/NOTE-WAP>, October 1998.
- [5] P. Mockapetris, *RFC-1034: Domain Names - Concepts and Facilities*, Internet Network Information, November 1987.
- [6] The Common Object Request Broker Architecture - version 2.1, *Corba Services Specification - Naming Services*, Object Management Group, December 1997.
- [7] W. Teong, T. Howes, F. Kille, *RFC-1777: Lightweight Directory Access Protocol*, Network Working Group, March 1995.

- [8] T. Howes, *RFC-2254: The String Representation of LDAP Search Filters*, Network Working Group, December 1997.
- [9] Identification Cards - Integrated Circuit(s) Cards with Contacts - Part 9, 7816-4: *Inter-Industry Commands for Interchange*, International Standard Organisation (ISO), 1998.
- [10] Identification Cards - Integrated Circuit(s) Cards with Contacts - Part 9, 7816-9: *Enhanced Inter-Industry Commands*, International Standard Organisation (ISO), 1998.
- [11] JavaCard 2.0, *Language Subset and Virtual Machine Specification*, Sun Microsystems Inc., JavaCard Forum, October 1997.
- [12] J.J. Vandewalle, E. Vetillard, *Developing Smart Card based Applications Using JavaCard*, Cardis'98, in proceedings of Third Smartcard Research and Advanced Application Conference, Springer-Verlag, Louvain-la-Neuve, Belgium, September 1998.
- [13] Digital Cellular Telecommunication System, *Specification of the Subscriber Identity Module (SIM)*, European Telecommunications Standards Institute (ETSI), July 1998.
- [14] Digital Cellular Telecommunication System, *GSM Public Land Mobile Network (PLMN)*, European Telecommunications Standards Institute (ETSI), October 1993.
- [15] Digital Cellular Telecommunication System, *Specification of the SIM Application programming Interface (SIM Toolkit)*, European Telecommunications Standards Institute (ETSI), July 1998.
- [16] Gemplus, *Understanding fundamentals of smartcard enabled security for Web and e-mail*, Gemplus White Paper, <http://www.gemplus.com>, September 1998.
- [17] *Platform for Privacy Preference (P3P)*, World Wide Web Consortium (W3C), July 1998.
- [18] Java Naming and Directory Interface, *Interface Specification*, JavaSoft, Sun Microsystems Inc., January 1998.
- [19] Java naming and Directory services, *Service Provider Interface Specification*, JavaSoft, Sun Microsystems Inc., January 1998.
- [20] M. Blaze, *High-Bandwidth Encryption with Low-Bandwidth Smartcards*, ftp://ftp.research.att.com/dist/mab/card_cipher.ps.
- [21] P. Biget, *The Vault, an Architecture for Smartcards to Gain Infinite Memory*, Cardis'98, in proceedings of Third Smartcard Research and Advanced Application Conference, Springer-Verlag, Louvain-la-Neuve, Belgium, September 1998.

Investigations of Power Analysis Attacks on Smartcards

Thomas S. Messerges
Motorola Labs
Motorola
tomas@ccrl.mot.com

Ezzy A. Dabbish
Motorola Labs
Motorola
dabbish@ccrl.mot.com

Robert H. Sloan¹
Dept. of EE and Computer Science
University of Illinois at Chicago
sloan@eecs.uic.edu

Abstract

This paper presents actual results from monitoring smartcard power signals and introduces techniques that help maximize such side-channel information. Adversaries will obviously choose attacks that maximize side-channel information, so it is very important that the strongest attacks be considered when designing defensive strategies. In this paper, power analysis techniques used to attack DES are reviewed and analyzed. The noise characteristics of the power signals are examined and an approach to model the signal to noise ratio is proposed. Test results from monitoring power signals are provided. Next, approaches to maximize the information content of the power signals are developed and tested. These results provide guidance for designing smartcard solutions that are secure against power analysis attacks.

1.0 Introduction

Cryptographers have traditionally analyzed cipher systems by modeling cryptographic algorithms as ideal mathematical objects. Conventional techniques such as differential [1] and linear [2] cryptanalysis are very useful for exploring weaknesses in algorithms represented as mathematical objects. These techniques, however, cannot address weaknesses in cryptographic algorithms that are due to a particular implementation in hardware. The realities of a physical implementation can be extremely difficult to control and often result in the leakage of side-channel information. Techniques developed in [3] show how surprisingly little side-channel information is required to break some common ciphers. Attacks have been proposed that use such information as timing measurements [4,5], power consumption [6], electromagnetic emissions [7] and faulty hardware [8,9]. Eliminating side-channel information or preventing it from being used to attack a secure system is an active area of research.

A growing number of researchers are beginning to address this issue of implementation. Systems that rely

on smartcards to provide security are of particular concern. In such systems, smartcards are often viewed as tamper-resistant devices that are secure against all but the most determined and well-financed attackers. However, this reliance on the tamperresistance of smartcards needs to be carefully scrutinized [10]. This becomes even more important in light of the recent attacks using side-channel information. It is often the case that important data stored on smartcards, such as a cryptographic key or an authentication certificate, needs to be kept secret to prevent counterfeiting of cards or the breaking of a system's security. Examples of smartcard systems that rely on the secrecy of the data resident on smartcards can be found in [11]. Such systems are potentially vulnerable because every time the smartcard system performs a computation using the secret data, side-channel information may be leaked.

Of all the previously mentioned sources of side-channel information, power measurements are perhaps the most difficult to control. Ultimately all calculations performed by a smartcard operate on logical ones or zeros. Current technological constraints result in different power consumptions when manipulating a logical one compared to manipulating a logical zero. An attacker of a smartcard can monitor such power differences and obtain useful side-channel information. Differential Power Analysis (DPA) [6] is a statistical approach to monitoring such power signals from a smartcard. Kocher et al. [6] claim one can monitor the actions of a single transistor within a smartcard using DPA. In [6], the authors outline a specific DPA attack against smartcards running the DES [12] algorithm.

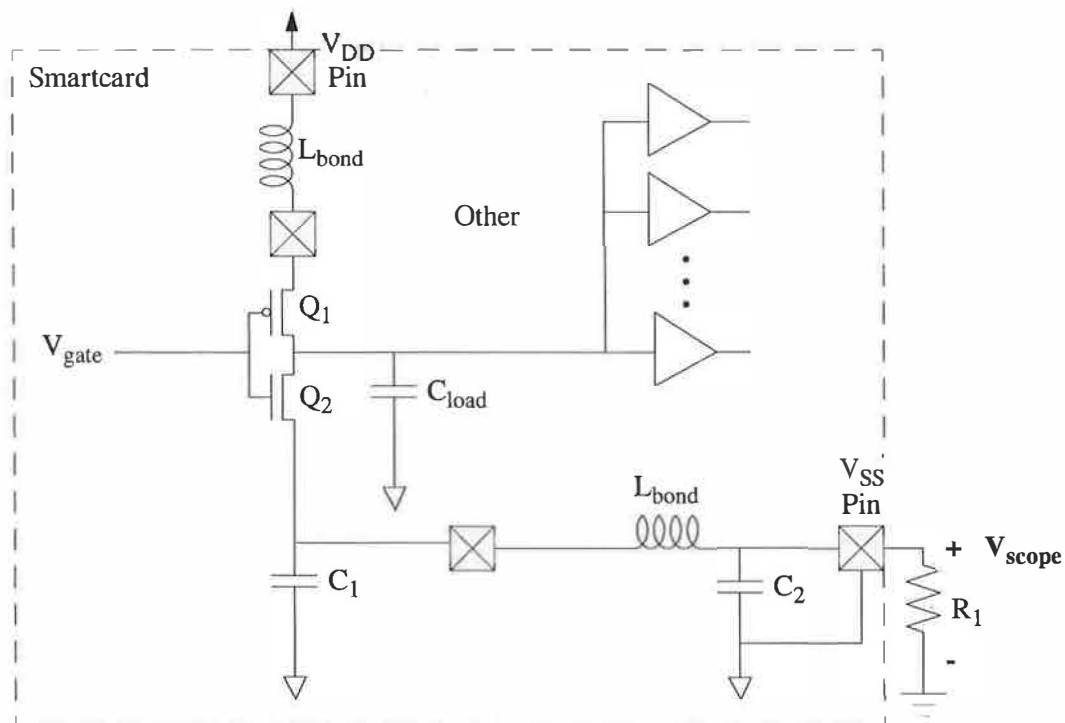
The purpose of this paper is to present actual results from monitoring smartcard power signals and to introduce techniques that help maximize such side-channel information. Whereas [3] showed how little side-channel information is required by an attacker, this paper takes the alternate approach and provides a first step towards showing how such information can be maximized. Adversaries will obviously choose attacks that maximize

1. Partially supported by NSF Grant CCR-9800070.

1.1 Smartcard Power Dissipation

power is dissipated when the circuit is clocked. This is known as dynamic power dissipation [13]. As V_{gate} changes from 0 to 5 volts, the transistors Q_1 and Q_2 are both conducting for a brief period causing current to flow from V_{dd} to ground. Also during this time, the capacitor C_{load} will be discharged (or charged) causing more (or less) current to flow through the V_{SS} pin.

Two types of information leakage from the data bus that have been observed are Hamming weight leakage and transition count leakage. Hamming weight information leaks when the dominant source of current is caused by the discharging of C_{load} . A situation where Hamming weight information leaks is when a precharged bus



All power signals in this report were measured across V_{scope} where R_1 was a 16 ohm resistor. A number of 8-bit microprocessor-based smartcards were examined and all produced results similar to those reported in this paper.

design is used. In this case, the number of zeros driven onto the precharged bus directly determines the amount of current that is being discharged. Transition count information leaks when the dominant source of current is due to the switching of the gates that are driven by the data bus. When the data bus changes state, many of the gates driven by the bus will briefly conduct current. Thus, the more bits that change state, the more power that is dissipated.

Experimental results that show transition count leakage from a typical 8-bit, microprocessor-based smartcard are given in Figure 2. In this figure, an 8-bit data byte from memory is transferred into a register. Initially, the bus contains the memory address, but after a clock pulse, the data from memory is put onto the bus. The magnitude of the voltage pulse is directly proportional to the number of bits that changed. Waveforms for different values of memory data are plotted on top of each other to effectively show this relationship. The difference in voltage between i transitions and $i+1$ transitions is about 6.5 mV. We observed similar plots for smartcards that leak Hamming weight information. The type of information that a particular smartcard leaks depends on the circuit design of the microprocessor and the type of operation being performed by the card. For instance, accessing EEPROM may yield different information than accessing RAM. Knowing which type of information is leaked will enable an adversary to optimize an attack strategy.

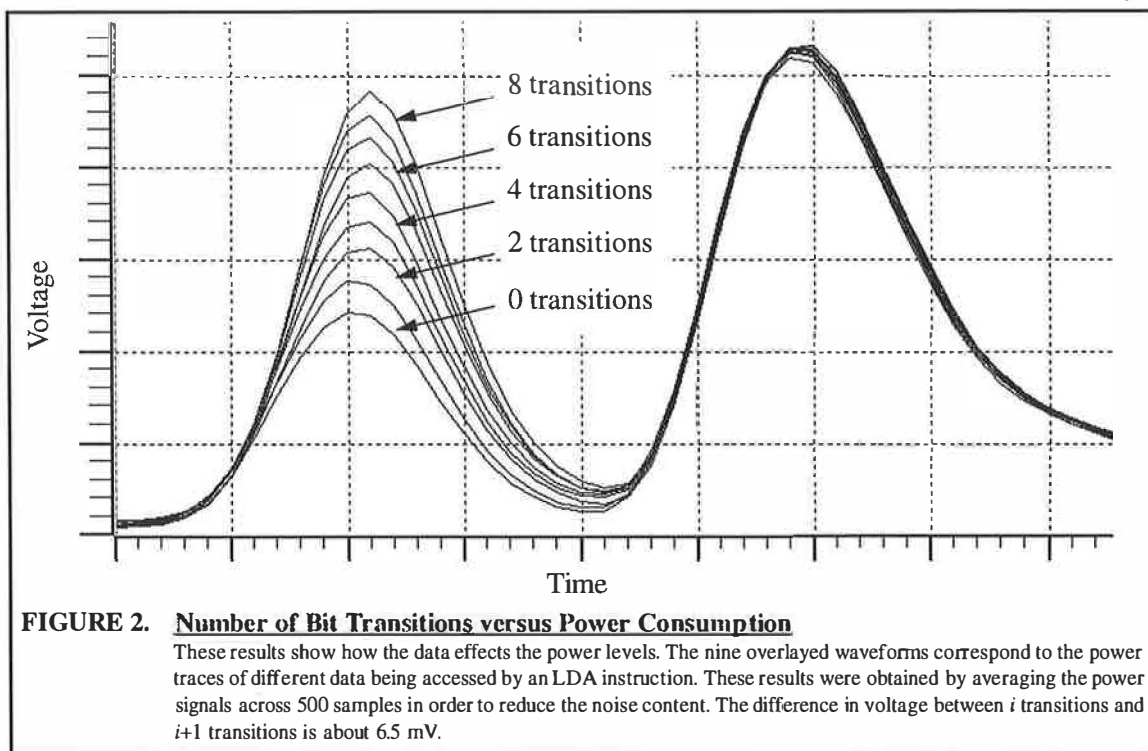
1.2 Simple Power Analysis (SPA)

An SPA attack, as described in [6], involves directly observing a system's power consumption. Different attacks are possible depending on the capabilities of the attacker. In some situations the attacker may be allowed to run only a single encryption or decryption operation. Other attackers may have unlimited access to the card. The most powerful attackers not only have unlimited access, but also have detailed knowledge of the software and hardware running in the card. If an attacker can determine where certain instructions are being executed, it can be relatively simple to extract useful information. For example, during the PC1 permutation in DES, we could determine the Hamming weight of each key byte by measuring the pulse height at the cycle of the instruction that accesses this data. In an 8-bit microprocessor, knowing the Hamming weight of all eight DES key bytes reduces the brute-force search space from 2^{56} to

$$\left[2\left(\frac{8^2}{128} + \frac{56^2}{128}\right)\right]^8 \approx 2^{45} \quad \text{or} \quad \left[2\left(\frac{1^2}{128} + \frac{7^2}{128} + \frac{21^2}{128} + \frac{35^2}{128}\right)\right]^8 \approx 2^{38}$$

keys depending on whether or not the parity bits are used. This was just an example; with algorithms using more key bits than DES or with triple-DES, knowing Hamming weight information alone does not help much with this type of brute-force attack.

A more powerful attack can result if the attacker can see Hamming weight information about the key bytes and also information about shifted versions of the key bytes.



In DES, such information can be leaked when shifting the C and D registers. In fact, given the weight of each byte for eight of the C and D shifts there is enough information to solve for the value of every key bit using the equation

$$A\vec{k} = \vec{w} \quad (1)$$

where \vec{w} is a 56×1 vector of Hamming weights, w_i ; \vec{k} is a 56×1 binary vector of the key bits, k_j ; and A is a 56×56 binary matrix such that A_{ij} is 1 if and only if weight w_i includes key bit k_j . Even algorithms with more than 56 key bits, such as triple-DES would be vulnerable to this attack.

If transition count information rather than Hamming weight information is available, an SPA attack can still be mounted, but it may be more difficult. The attacker would need to know the contents of the data bus before or after the data being sought is accessed. Many times this data is easy to determine because it is a fixed address or an instruction opcode. Attackers with access to the code can easily get this data and set up equations similar to Equation (1). Other less knowledgeable attackers may need to resort to trial and error to determine the correct equations. Due to the limited number of possibilities such an approach is reasonable.

Our experiments confirmed that poor implementations of DES will almost always be vulnerable to SPA attacks. Shifting the key bytes or the use of conditional branches to test bit values can be especially vulnerable. Also, if the code or portions of the code run in variable time, power analysis could be used to enable a timing attack [4]. Fortunately, implementors of cryptographic algorithms have known about these issues for a number of years. Kocher et al. [6] reported that it is not particularly difficult to build SPA-resistant devices. However, the attackers keep getting smarter so further research and vigilance will always be necessary.

1.3 Differential Power Analysis (DPA)

A DPA attack, described in [6] and reviewed here, is more powerful than an SPA attack because the attacker does not need to know as many details about how the algorithm was implemented. The technique also gains strength by using statistical analysis to help recover side-channel information. The objective of the DPA attacks described in this paper is to determine the secret key used by a smartcard running the DES algorithm. These techniques can also be generalized to attack other similar cryptographic algorithms.

A DPA attack begins by running the encryption algorithm for N random values of plain-text input. For each of the N plain-text inputs, PTI_i , a discrete time power signal, S_{ij} , is collected and the corresponding cipher-text output, CTO_i , may also be collected. The power signal S_{ij} is a sampled version of the power consumed during the portion of the algorithm that is being attacked. The i index corresponds to the PTI_i that produced the signal and the j index corresponds to the time of the sample. The S_{ij} are split into two sets using a partitioning function, $D(\cdot, \cdot, \cdot)$:

$$\begin{aligned} S_0 &= \{S_{ij} | D(\cdot, \cdot, \cdot) = 0\} \\ S_1 &= \{S_{ij} | D(\cdot, \cdot, \cdot) = 1\} \end{aligned} \quad (2)$$

The next step is to compute the average power signal for each set:

$$\begin{aligned} A_0[j] &= \frac{1}{|S_0|} \sum_{S_{ij} \in S_0} S_{ij} \\ A_1[j] &= \frac{1}{|S_1|} \sum_{S_{ij} \in S_1} S_{ij} \end{aligned} \quad (3)$$

where $|S_0| + |S_1| = N$. By subtracting the two averages a discrete time DPA bias signal, $T[j]$, is obtained:

$$T[j] = A_0[j] - A_1[j] \quad (4)$$

Selecting an appropriate D function will result in a DPA bias signal that an attacker can use to verify guesses of the secret key. An example of such a D function is as follows:

$$D(C_1, C_6, K_{16}) = C_1 \oplus \text{SBOX1}(C_6 \oplus K_{16}) \quad (5)$$

where

C_1 = the 1 bit of CTO_i that is XOR'ed with bit #1 of S-box #1

C_6 = the 6 bits of CTO_i that are XOR'ed with subkey K_{16}

K_{16} = 6 bits of the 16th round subkey feeding into S-box #1

$\text{SBOX1}(x)$ = a function returning bit #1 resulting from looking up x in S-box #1

The D function of Equation (5) is chosen because at some point during a DES implementation, the software needs to compute the value of this bit. When this occurs or anytime data containing this bit is manipulated, there will be a slight difference in the amount of power dissipated depending on whether this bit is a zero or a one. If

this difference is ϵ , and the instructions manipulating the D bit occurs at times j^* , the following expected difference in power equation results:

$$E[S_{ij}|D(\cdot, \cdot, \cdot) = 0] - E[S_{ij}|D(\cdot, \cdot, \cdot) = 1] = \epsilon \quad (6)$$

for $j = j^*$

When j is not equal to j^* , the smartcard is manipulating bits other than the D bit, and the power dissipation is independent of the D bit:

$$\begin{aligned} E[S_{ij}|D(\cdot, \cdot, \cdot) = 0] - E[S_{ij}|D(\cdot, \cdot, \cdot) = 1] \\ = E[S_{ij}] - E[S_{ij}] = 0 \quad \text{for } j \neq j^* \end{aligned} \quad (7)$$

As the number N of PTI inputs is increased, Equation (4), converges to the expectation equation:

$$\begin{aligned} \lim_{N \rightarrow \infty} T[j] &= E[S_{ij}|D(\cdot, \cdot, \cdot) = 0] \\ &\quad - E[S_{ij}|D(\cdot, \cdot, \cdot) = 1] \\ &= E[S_{ij}] - E[S_{ij}] = 0 \quad \forall j \end{aligned} \quad (8)$$

Thus, a review of Equations (6), (7) and (8) shows that if enough PTI samples are used, $T[j]$ will show power

biases of ϵ at times j^* , and will converge to zero all other times. Due to small statistical biases in the S-box outputs, the assumption of independence in Equation (7) is not completely true. In reality, $T[j]$ will not always converge to zero; however the largest biases will occur at times j^* .

One input to the D function was K_{16} , six bits of the subkey. The attacker does not know these bits, but can use brute force and try all 2^6 possible values. For each guess, the attacker constructs a new partition for the power signatures and gets a new bias signal, $T[j]$. If the proper D function was chosen, the bias signal will show spikes whenever the D bit was manipulated. If the D function was not chosen correctly (i.e., the wrong subkey bits were guessed), then the resulting $T[j]$ will not show any biases. Using this approach, an attacker can determine the six subkey inputs to S-box #1 at round 16 of DES. Repeating this approach for the seven other DES S-boxes allows the attacker to learn the entire round 16 subkey (48 bits). The remaining 8 bits can be found by brute force or by successively applying this approach backwards to previous rounds. Figure 3 compares the bias signal for a correctly chosen key to that of an incorrectly chosen key. The signal for the incorrect key shows a few small biases, but the correct key has biases that are twice as large, so it is easily recognized. We created the signals in Figure 3 using $N = 1300$.

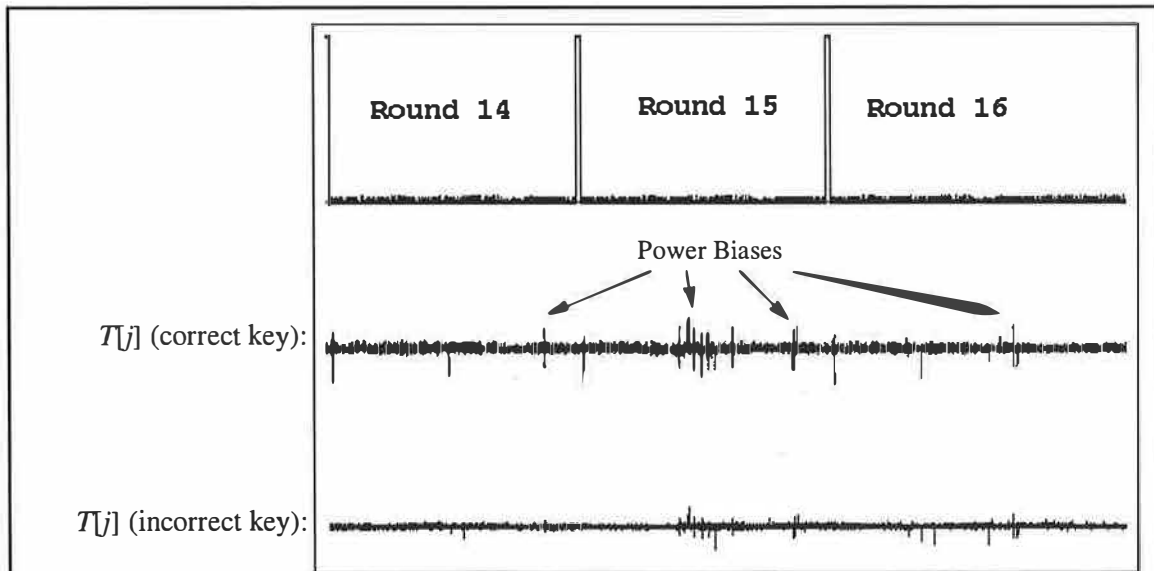


FIGURE 3. DPA Result Showing Power Bias Signals for Correct Key and Incorrect Key

The power biases are about 6.5 mV for the correct key and about half the size for the incorrect key. The voltage SNR for the correct key is 17.3. A total of $N=1300$ power signals were used to generate the above plots.

2.0 Noise in Power Analysis Attacks

There is a rich history of methods to reduce noise in electrical circuits. Many good textbooks have been written on this topic (e.g., [14]). Techniques for noise reduction are particularly important when trying to measure side-channel emanations because the magnitude of these signals can be extremely small. The DPA attack uses averaging to reduce noise, but it is important to investigate other strategies that may lead to further reductions in the amount of noise. Experts quoted in [15] report that one way to prevent a power analysis attack is to mask the side-channel information with random calculations that increase the measurement noise. A good understanding of the achievable noise levels using typical electronic measuring techniques is needed to evaluate the effectiveness of such a solution.

2.1 Noise Characteristics

There are four types of noise present when performing power analysis of smartcards: external, intrinsic, quantization and algorithmic. External noise is generated by an external source and coupled into the smartcard. Intrinsic noise is due to the random movement of charge carriers within conductors. Quantization noise is due to the quantizer in the A/D that is used to sample the power signals. Algorithmic noise is due to the randomness of the data bytes being processed by the smartcard. Intrinsic and quantization noise are small when compared to present day side-channel signals, but will likely play a larger role

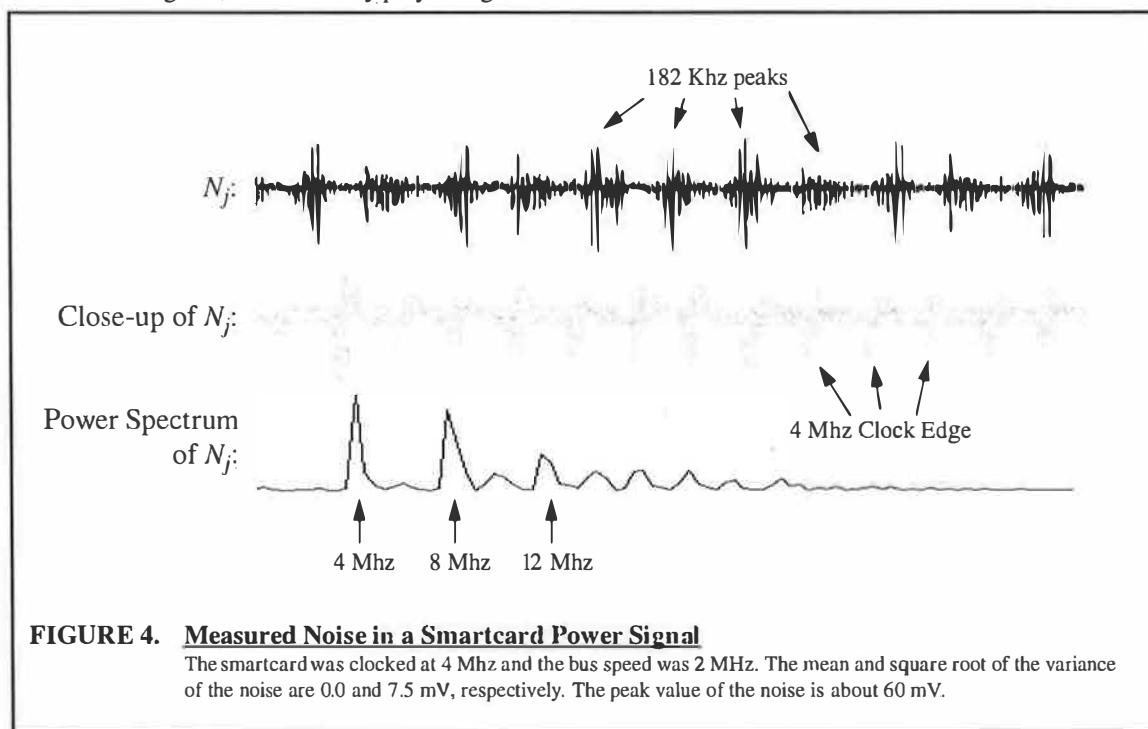
as future designs minimize side-channel signals. External noise can be reduced through careful use of measuring equipment, good circuit design practices and filtering. Algorithmic noise can be reduced if an attack strategy can use unbiased random data that can be averaged out.

Figure 4 illustrates a sample of the noise measured in a power analysis experiment. The top waveform, N_j , was determined using the following equation:

$$N_j = E[S_j] - S_j$$

where the value of $E[S_j]$ was estimated by averaging the power traces obtained when encrypting the same input data 5,000 times and S_j is one of the traces. Clearly any deviation of N_j from zero should be considered noise. The noise shown in Figure 4 has a slow beat frequency around 182 KHz that could be due to external coupling from test equipment or more likely due to the superposition of the digital oscilloscope's sampling frequency and the smartcard's clock frequency. The power spectrum of the noise also reveals significant spikes at the smartcard's clock frequency and its harmonics. A close-up of the noise reveals that a significant amount of the noise energy is concentrated at the edges of the smartcard's 4 Mhz clock edges.

The noise signal in Figure 4 does not contain algorithmic noise since the data being encrypted was kept constant. The effect of algorithmic noise would be to cause spikes



to appear in the portion of the signal where the data is being processed. These spikes could be used by an attacker to mark locations in the power trace that are data dependent. Algorithm noise can be modeled as shot noise using a generalized Poisson random process [16]:

$$x(j) = \sum_i c_i F(j - j_i)$$

In this case, c_i is a random variable (r.v.) corresponding to the Hamming weight of the data being processed, j_i is a Poisson r.v. that models the seemingly random locations of the data dependencies and F would be the shape of the bias signal. If random data is used, then c_i would have a binomial distribution. An accurate model of the various noise components would be very useful for simulating the effectiveness of DPA attacks and solutions.

2.2 Filtering the Noise

Filtering strategies can be used to reduce the noise shown in Figure 4, but care needs to be taken so that the components necessary to create a power bias signal are not affected. The algorithmic noise can be reduced by averaging using unbiased random data. Filtering will not work on algorithmic noise because any filter would also reduce the desired bias signal. If the noise in Figure 4 is assumed to be white noise, then the optimal filter that maximizes the signal-to-noise ratio (SNR) is the matched filter. We designed and tested such a filter for the bias signal shown in Figure 3. The original voltage SNR was 17.3. After using the matched filter, the SNR was increased to 23.2. This gives some indication of how much an attacker with a perfect matched filter can improve the SNR.

2.3 The Effect of Averaging

A DPA attack is successful because averaging reduces noise energy, thus revealing any concentrations of signal energy that occur at constant positions in time. To see the effect of averaging on the noise refer back to Equation (3). Let the average variance of S_{ij} be σ^2 and assume S_{ij} is independent of S_{kj} when $i \neq k$. If the N signals were equally split between sets S_0 and S_1 , then the variance of A_0 and A_1 is $2\sigma^2/N$. Therefore, when A_0 and A_1 are combined the resulting DPA bias signal has a variance of $4\sigma^2/N$.

The averaging effect on the signal was shown in Equation (6) to produce a bias spike of size ϵ . The D function, proposed in Equation (5), has the effect of fixing one bit of the data being processed. If a data word

is m bits wide, then the remaining unfixed bits in the word have an average Hamming weight of $(m-1)/2$ with a variance (i.e., algorithmic noise) of $(m-1)/4$. Averaging over N samples has a similar effect on this signal variance as it did on the noise variance. Assuming independence of the different sources of noise yields the following signal and noise parameters:

$$\begin{aligned} \text{noise: } E[T[j] | (j \neq j^*)] &= 0 \\ \text{var} [T[j] | (j \neq j^*)] &= \frac{4\sigma^2 + \alpha m \epsilon^2}{N} \end{aligned} \quad (9)$$

$$\begin{aligned} \text{signal: } E[T[j^*]] &= \epsilon \\ \text{var} [T[j^*]] &= \frac{4\sigma^2 + (m-1)\epsilon^2}{N} \end{aligned}$$

In the variance of the noise, the term α is the percentage of $T[j]$ that is dependent on the data being encrypted, thus the percentage of algorithmic noise. The success of a DPA attack depends on distinguishing the signal from noise, so the final voltage SNR is:

$$\text{SNR} = \frac{\sqrt{N}\epsilon}{\sqrt{8\sigma^2 + \epsilon^2(\alpha m + m - 1)}} \quad (10)$$

Equation (10) can be checked with previously measured experimental results. Setting $\sigma = 7.5$ mV (from Figure 4), $\epsilon = 6.5$ mV (from Figure 2), $m = 8$, $N = 1000$, and $\alpha \approx 0$ yields $\text{SNR} = 7.5$. Experimental results using these parameters showed a SNR between 7 and 10, thus confirming that Equation (10) is a valid approximation.

3.0 Maximizing DPA Bias Signal

One way to increase the SNR in Equation (10) is to increase ϵ . The value of ϵ is dependent on the number of bits output by the D function. In Equation (5), D outputs only one bit, but in general the D function could output d bits. In this general case $\epsilon = dl$, where l is a constant equal to the voltage difference seen between two data words with Hamming weight i and $i+1$. Figure 2 showed that these differences can be considered approximately equal, for all i .

When performing the differential power analysis there would be three sets used for partitioning:

$$S_0 = \left\{ S_{ij} | D(., ., .) = 0^d \right\}$$

$$S_1 = \left\{ S_{ij} | D(., ., .) = 1^d \right\}$$

$$S_2 = \left\{ S_{ij} | S_{ij} \notin S_0, S_1 \right\}$$

The other DPA equations would all remain the same and the power signals in set S_2 would not be used. The following signal and noise equations would result:

$$\begin{aligned} \text{noise: } E[T[j] | (j \neq j^*)] &= 0 \\ \text{var}[T[j] | (j \neq j^*)] &= \frac{4\sigma^2 + \alpha m l^2}{N} \\ \text{signal: } E[T[j^*]] &= dl \\ \text{var}[T[j^*]] &= \frac{4\sigma^2 + (m-d)l^2}{N} \end{aligned} \quad (11)$$

$$\text{SNR} = \frac{\sqrt{N}dl}{\sqrt{8\sigma^2 + l^2(\alpha m + m - 1)}}$$

3.1 4-bit DPA Attack on DES

In DES, a 4-bit D function based on the four bits output from an S-box can be used to partition the sets S_0 and S_1 . Equation (5) can be modified to output four bits:

$$D(C_6, K_{16}) = \text{SBOX1}_4(C_6 \oplus K_{16}) \quad (12)$$

where $\text{SBOX1}_4(x)$ returns all four bits resulting from looking up x in SBOX #1. Similar D functions would be used for the other S-boxes to enable all the key bits to be eventually determined.

The question arises as to whether Equation (12) will result in partitions that are unbiased enough so that the incorrect choices of K_{16} will result in random power signals that average to zero. A simulation was written to see what type of biases could be expected. The results for 4-bit DPA and 1-bit DPA using random PTI inputs and a typical key are graphed in Figure 5. The graph shows that the expected Hamming weight biases of the S-box #1 lookup for each of the 64 possible key guesses. The correct key guess clearly shows the most bias; however in the case of 4-bit DPA there is a case where an incorrect key guess has the same magnitude bias as the correct key. In this case, if the attacker did not know the sign of the expected bias, a very small brute-force search might be needed.

3.2 Multiple Bit DPA

In general, other multiple bit D functions can be defined. In software implementations, the S-box lookups are performed using a load instruction from a table storing all

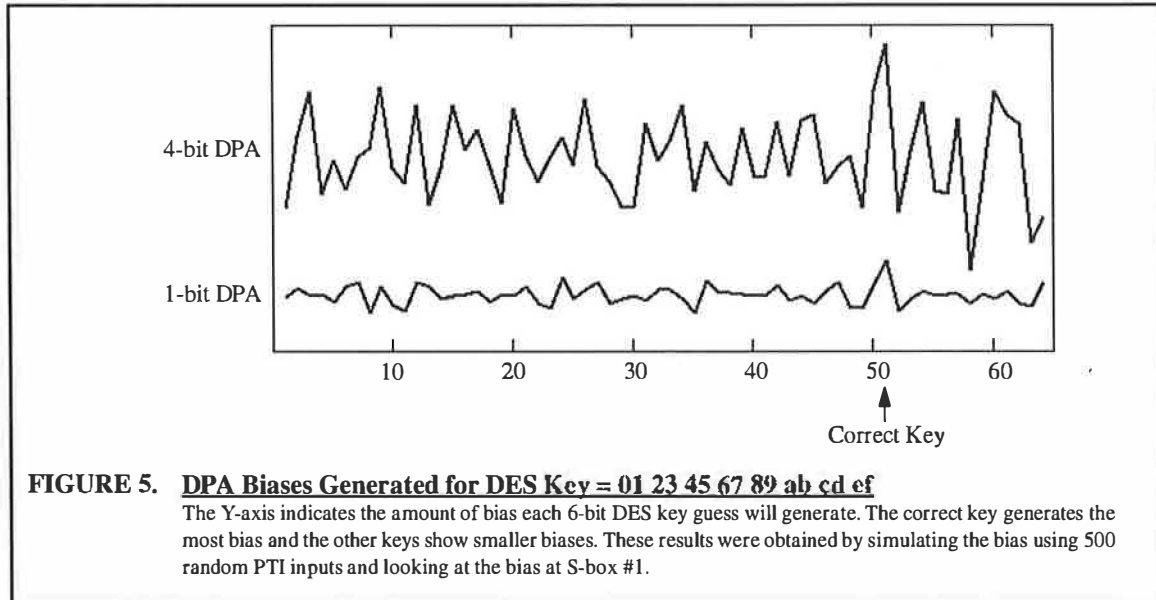


TABLE 1:

d :	1	2	3	4	5	6	7	8
N_d :	N	$0.5N$	$0.44N$	$0.5N$	$0.64N$	$0.88N$	$1.3N$	$2N$

TABLE 2:

Attack Type:	1-bit DPA	4-bit DPA	8-bit DPA	Address DPA
Signal Level:	9.3 mV	38.5 mV	79.5 mV	74.4 mV

the S-box data. To compress the table size, the four bit S-box data is frequently stored in pairs of two or more, depending on the word size of the processor. If the attacker knows how the S-box data is packed into this table, it would be possible to maximize the bit biases even further. For example, power signals with S-box lookups that yield many zeros can be separated from signals with S-box lookups containing many ones. Again, the number of bits that can be biased will proportionally increase the SNR.

Another way to mount an attack would be to partition S_0 and S_1 based on the addresses used for the S-box lookups rather than the data. The attacker forms two partitions, one that maximizes the number of address bus transitions and one that minimizes the number of address bus transitions. Power signatures from both partitions are averaged and then subtracted. If the address bus is larger than the data bus, even larger biases could be achieved.

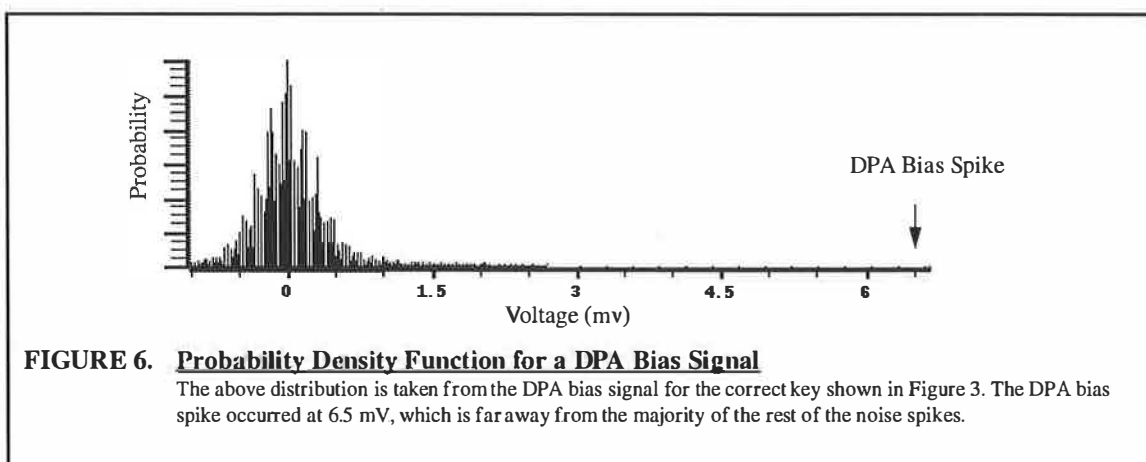
When mounting a d -bit DPA attack the attacker may need to use more power signals. After all, an average of $1/2^{1-d}$ of the power signals are placed in partition S_2 and are unused. This suggests that an attacker cannot increase d too much without requiring an excessive number of power signals. An attacker running 1-bit DPA using N signals would need $N_d = 2^{d-1}N/d^2$ power signals to maintain the same SNR when running d -bit DPA. Table 1 shows that for small values of d actually fewer power signals are necessary and with 8-bit DPA only twice as many signals are needed. The main advantage of

d -bit DPA is that the resulting signal levels are magnified d times.

Experiments were run on a smartcard to confirm the effectiveness of 4-bit, 8-bit and address-bit DPA. The resulting signal levels are shown in Table 2. In these experiments, the 8-bit DPA attack was only able to create an average bias of 6.0 bits because the S-box data was never all zeros or all ones. The address-bit DPA had similar problems so only an average bias of 6.5 bits was achieved. As can be seen the signal levels for these multiple bit DPA attacks are much stronger. It is clear that countermeasures to DPA attacks need to consider all these more powerful attacks.

3.3 Hiding the DPA Bias Spike

One suggested solution to prevent DPA attacks is to add random calculations that increase the noise level enough to make the DPA bias spikes undetectable. The results presented in this paper give some indication of how much noise needs to be added. The main goal is to add enough random noise to stop an attack, but to add minimal overhead. A rough approximation of the noise necessary can be found by looking at the probability density function (p.d.f.) of the DPA bias signal plotted in Figure 6. The DPA spike obviously stands out from the noise because it is very improbable that it was generated by noise. Ideally, the p.d.f. of the bias signal would stretch out to cover the DPA spike or the DPA spike would be reduced so that it is closer to the noise. One



suggested design criterion is that the DPA bias spike have equal amounts of noise distributed above and below it. This means the median of the noise signal would be at the level of the DPA bias spike. If the noise were Gaussian, the median would be when the SNR=0.67. One could then use Equation (11) to determine what amount of noise is necessary to prevent the desired level of attack. Of course a different strategy would need to be used if the noise p.d.f. was not normal or the noise around the DPA spike was non-stationary. The fact that the DPA bias spikes and most of the noise energy occurs near the clock edges can also be considered. Less noise may be necessary because the DPA spikes occur in these areas of high noise energy.

4.0 Future Work

Future research in this area will investigate power analysis attacks on hardware encryption devices and public-key cryptosystems. Preliminary work suggests that such systems will also be vulnerable, but specific attacks have not yet been evaluated. The analysis of more symmetric-key algorithms is also an important topic that will be investigated. Ways to design and implement new algorithms that are not vulnerable to these attacks will be researched. The solutions to prevent these types of side-channel attacks need to be carefully scrutinized. Comprehensive analysis techniques, testing procedures and more advanced modeling methods will be developed.

5.0 Conclusions

Attacks that monitor side-channel information and in particular power analysis attacks have recently been getting much attention by experts in the smartcard industry. The results presented in this paper confirm that power analysis attacks can be quite powerful and need to be addressed. Ways an attacker might maximize the side-channel signals have been investigated and were found to be very effective. Solutions to prevent DPA attacks need to consider these advanced attacks in order to provide the maximum amount of security. Understanding the noise characteristics of the power signals is also very important. The experimental and theoretical results presented in this paper hopefully will be helpful for modeling the problem and designing the solutions.

Much of the concern in industry is how to safeguard existing products. These products are mostly software implementations similar to the ones examined in this paper. The new attacks and analysis results presented in this paper are applicable to these designs and to future designs and hardware implementations. When creating a

new hardware encryption circuit or algorithm implementation, a designer needs to ask the question: "What is the strongest potential power analysis attack that can be mounted by an attacker?" With the answer to this in mind, a designer can successfully protect against power analysis attacks.

6.0 References

- [1] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptography*, Vol. 4, No. 1, 1991, pp. 3-72.
- [2] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," in Proceedings of *Advances in Cryptology—Eurocrypt '93*, Springer-Verlag, 1994, pp. 386-397.
- [3] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side Channel Cryptanalysis of Product Ciphers," in Proceedings of *ESORICS '98*, Springer-Verlag, September 1998, pp. 97-110.
- [4] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in Proceedings of *Advances in Cryptology—CRYPTO '96*, Springer-Verlag, 1996, pp. 104-113.
- [5] J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestré, J. J. Quisquater and J. L. Willems, "A Practical Implementation of the Timing Attack," in Proceedings of *CARDIS 1998*, September 1998.
- [6] P. Kocher, J. Jaffe, and B. Jun, "Introduction to Differential Power Analysis and Related Attacks," <http://www.cryptography.com/dpa/technical>, 1998.
- [7] W. van Eck, "Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk," *Computers and Security*, v. 4, 1985, pp. 269-286.
- [8] D. Boneh and R. A. Demillo and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in Proceedings of *Advances in Cryptology—Eurocrypt '97*, Springer-Verlag, 1997, pp. 37-51.
- [9] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," in Proceedings of *Advances in Cryptology—CRYPTO '97*, Springer-Verlag, 1997, pp. 513-525.

- [10] R. Anderson and M. Kuhn, "Tamper Resistance—A Cautionary Note," *The Second USENIX Workshop on Electronic Commerce Proceedings*, 1996, pp. 1-11.
- [11] *Cardtech/Securtech '98 - Conference Proceedings, Volume II: Applications*, 1998.
- [12] ANSI X.392, "American National Standard for Data Encryption Algorithm (DEA)," American Standards Institute, 1981.
- [13] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley Publishing Company, 1993.
- [14] E. R. Davies, *Electronics Noise and Signal Recovery*, San Diego, CA: Academic Press Inc., 1993.
- [15] P. Wayner, "Code Breaker Cracks Smart Cards' Digital Safe," *New York Times*, June 22, 1998, pp. C1. (or <http://www.nytimes.com/library/tech/98/06/biztech/articles/22card.html>).
- [16] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, New York: McGraw-Hill, Inc. 1991.

Risks and Potentials of using EMV for Internet Payments

Els Van Herreweghen

IBM Zurich Research Laboratory, Ruschlikon, Switzerland

Email: evh@zurich.ibm.com

Uta Wille*

Jelmoli Information Systems AG, Zurich, Switzerland

Email: wille_u@jelmoli.ch

Abstract

Existing payment smartcards developed for traditional point-of-sale transactions are being considered for use in Internet transactions. Such solutions have been suggested as alternatives to using payment protocols more specifically designed for Internet payments (such as SET [8]) but often lacking smartcard support. In this paper, we analyze EMV'96 [7], a representative example of an existing payment smartcard specification. We investigate which security requirements for an Internet payment system can and cannot be met when using EMV for Internet payments. We suggest possible modifications that can enhance the security of an Internet payment scheme based on EMV.

1. Introduction

With the growing use of the Internet for commercial transactions, there has been much effort in developing systems and protocols for securing payments on the Internet. A prominent example of such a protocol is the Secure Electronic Transaction (SET, [8]) protocol. It is, however, not designed with smartcard support in mind. Current implementations require the customer to make SET transactions from a fixed, trusted personal computer. A secure SET implementation on a smartcard for use with public (and untrusted) terminals would require the smartcard to store the user's account data and cryptographic keys as well as the SET client implementation, which is not feasible with current smartcard technology.

The lack of portability of Internet-specific systems such as SET has caused the payment industry to look at the possibility of using existing debit and credit payment smartcards for Internet payments. A standard in this area is the EMV'96 Specification [7], which describes the functionality required by such smartcard-based payment systems.

The objective of this paper is to discuss the potentials and restrictions of using EMV payment cards for debit and credit payments over the Internet. In Section 2 we formulate a set of general security requirements for smartcard-based debit and credit Internet payments. After summarizing EMV'96 security mechanisms in Section 3, we analyze in Section 4 the security properties of using EMV 'as is' for Internet payments, by checking the resulting protocols against the formulated requirements. Since the Internet scenario differs from the scenario assumed by EMV'96, these protocols show a number of vulnerabilities. In Section 5, we finally discuss mechanisms to increase the security of using EMV in the Internet scenario.

2. Model and Security Requirements for Smartcard Internet Payments

Our model of a generic Internet payment system (Figure 1) consists of a customer and a merchant who exchange money for goods or receipts, and at least one financial institution linking electronic payments to the transfer of "real money" [1].

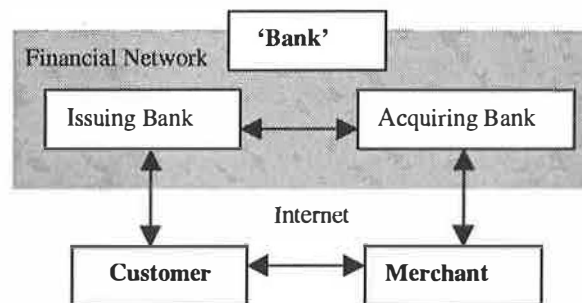


Figure 1. Payment Model.

Customer and merchant communicate over an open network (the Internet) with each other and with their banks (issuing bank and acquiring bank, respectively).

* This paper reports on work done at the IBM Zurich Research Laboratory, Ruschlikon, Switzerland.

During a transaction, actual connectivity may be limited to certain subsets of players. In a typical online purchase scenario, the customer only has a connection to the merchant, and communicates indirectly with his issuing bank (e.g., through an authorization message sent to the merchant and forwarded by the merchant to acquiring and issuing banks). The underlying communication model, however, does not influence the security requirements stated in the following.

Before formulating security requirements for a payment transaction, we need to make a number of assumptions about trust relations and liability distributions between the parties involved:

- A1. Issuer and acquirer enjoy some degree of mutual trust and share an infrastructure for secure communication. This allows us to join their security requirements into “bank” requirements.
- A2. Money transfers between accounts are traceable. This gives the user some assurance of refund in case of fraud by hackers or bank insiders.
- A3. A contract determines the business, trust, responsibility, and liability relationships between the merchant and the bank. The contract especially defines valid payments by specifying the requirements to be fulfilled to provide the merchant with a payment guarantee.
- A4. A contract determines the business, trust, responsibility, and liability relationships between the customer and the issuing bank. The contract defines what the bank considers proofs of payment by the customer and specifies the requirements for liability and disputability.
- A5. The customer (user) can trust critical parts of his or her system to enable secure authorization of a transaction. In the specific case of the user’s payment instrument being a smartcard authorizing the payment on the user’s behalf, the user interacts with the card reader (or electronic wallet) by verifying output (e.g., transaction amount, merchant ID) on its display, and by entering data (e.g., PIN-code) on a keyboard or PIN-pad. The user can trust that:
 - The correct transaction data are displayed;
 - Secret data such as a PIN-code entered by the user is not exposed or intercepted.

We now list the requirements on a payment protocol in the above model. Requirements R1 to R7 apply to electronic payment protocols in general. Requirement R8 is related to controlling access to the customer’s payment instrument and is treated with a special focus on the use of smartcards.

A number of requirements deal with proof of *authorization of the transaction by an authorizing party to a verifying party*. This is achieved by an authorization message containing a non-forgeable cryptographic proof of authentication by the authorizing party of critical transaction-related data, satisfying the following properties:

- The verifying party can verify authenticity and integrity of the critical data in the authorization message, and can verify that the data originated from the authorizing party;
- The message cannot be used to authorize another transaction (non-replayable); nor can it be used in any other way by an attacker to falsely authorize another transaction on behalf of the customer. The last requirement applies to schemes where secret authorization data (such as a PIN) is sent to the bank for verification. In such cases, this requirement translates into the requirement that this data be confidentiality-protected (encrypted) during transfer from card to bank.

As in [3], we furthermore distinguish between *weak* and *undeniable* proofs of authorization. A weak proof (e.g., shared-key based EMV Application Cryptogram, Section 3.3) cannot serve as a proof for third parties while an undeniable proof (based on a digital signature) provides non-repudiation and therefore can be used in the case of disputes. Based on these notions we formulate the following security requirements for a payment protocol:

- R1. **Authorization customer to bank.** The bank possesses a payment authorization from the customer before debiting the customer’s account.
- R2. **Authorization merchant to bank.** The bank only authorizes a payment to a merchant if the corresponding transaction has been authorized by that merchant.
- R3. **Payment guarantee for merchant** before delivery of goods. This is achieved by either of:
 - i. *Authorization of the transaction by the bank;*
 - ii. *Authorization of the transaction by the customer,* where the bank guarantees customer-approved transactions (see assumption A3).
- R4. **Authentication and certification of merchant to customer.** The customer has a minimum of authenticated and certified information about the merchant s/he makes a payment to.
- R5. **Payment receipt for customer.** After completion of the payment, the customer possesses a proof that the payment was successful. This can either be:

- i. Explicit *payment receipt from the merchant*;
- ii. *Payment receipt from the bank*.

It is sometimes assumed that a receipt can be replaced by a statement of account [3].

- R6. **Atomicity of payments.** No party can benefit from an interrupted protocol run.
- R7. **Privacy, anonymity.** The customer may require privacy of order and payment information and possibly anonymity (from eavesdroppers and eventually from merchants and/or banks).
- R8. **Cardholder authorization.** The customer's payment system should be protected against unauthorized use. In the case of smartcard-based payments, unauthorized use of the card should be protected against (e.g. through use of a PIN). As mentioned in A5, the customer also needs to trust at least the terminal (or electronic wallet) s/he's using in conjunction with the smartcard.

3. Security Mechanisms Provided by EMV

This section gives an overview of EMV'96 security mechanisms securing transaction flows. Mechanisms such as card and terminal risk management are not discussed here. For a detailed description of security mechanisms provided by EMV'96 we refer to [7].

Figure 2 shows the general EMV POS scenario of an IC (Integrated Circuit) terminal interacting with an IC card, with the human user presenting the card, and with the bank. (The actual EMV functionality for authorizing transactions resides with the issuing bank. Here we make abstraction of the distinction between the issuing bank and the merchant's acquiring bank.)

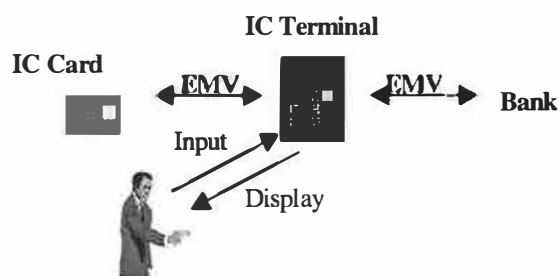


Figure 2. The EMV POS Scenario

- Terminal-card interaction consists of EMV commands issued by the terminal and card responses;
- Interaction between terminal and bank consists of the exchange of authorization requests and responses, often over a telephone connection;
- Interaction between terminal and human user consists of output to the user via the terminal

display, and input by the user authorizing the transaction (such as a PIN-code).

EMV uses both asymmetric (public-key) and symmetric (shared-key) security mechanisms:

- Asymmetric security mechanisms authenticate the card as a valid card to the terminal;
- Symmetric security mechanisms generate and verify transaction cryptograms (essentially Message Authentication Codes, MACs) based on a key k shared between card and (issuer) bank.

The full set of security mechanisms is shown in Figure 3 which is taken from a transaction flow example in [7]. For reasons of simplicity, we make abstraction of most options and variants of the security mechanisms and focus on showing the maximum security features that can be achieved by an EMV compliant transaction.

3.1. Public-key based authentication of IC card to IC terminal

The first four messages exchanged implement the *Dynamic Data Authentication* (DDA) authenticating the card to the terminal using a public-key based challenge-response protocol. The READ_RECORD command returns the necessary Certification Authority (CA) identifier and public-key certificates needed by the terminal to authenticate the card's public key in CERT_C. CERT_C is certified by the issuer and can be verified using the issuer's public key in CERT_I, which in turn is certified by the CA and can be verified using the CA's public key known to the terminal. The actual challenge-response authentication is then executed by the terminal issuing an INTERNAL_AUTHENTICATE command containing authentication-related data (ARD), and the card responding with a signature over this data using its private signature key.

For cards without digital signature capability, EMV also provides the *Static Data Authentication* mechanism using static card data signed by the Issuer.

3.2. Cardholder Verification

EMV supports online (PIN sent to and verified by the bank) and offline (PIN verified by the card) PIN verification; the exact method supported by the card is read by the terminal with the initial READ_RECORD. Offline PIN verification is executed by the terminal issuing the VERIFY command, containing the PIN data entered by the user; the card's response indicates success or failure. The response is not cryptographically authenticated.

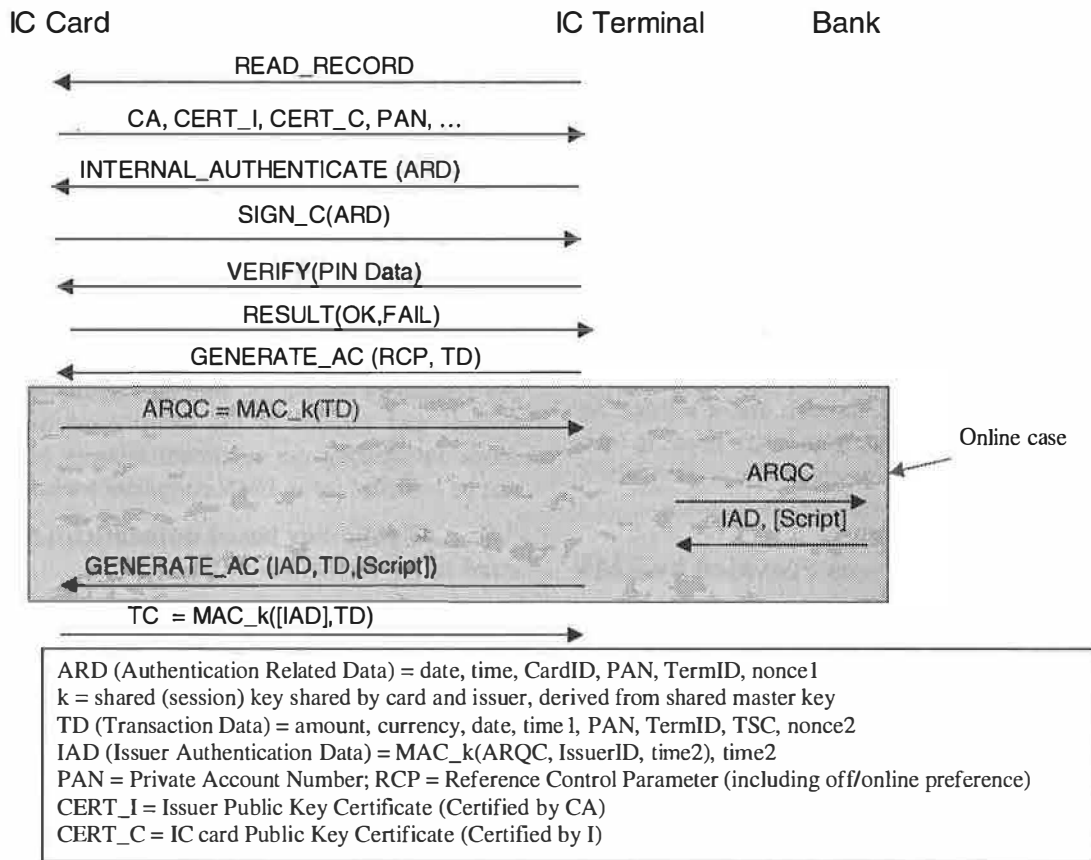


Figure 3. Model EMV Transaction Flow

3.3. Shared-key based application cryptograms and off- or online processing

The GENERATE_AC command, including Transaction Data (TD), triggers the card to produce a cryptogram that can be verified by the issuer. If both card and terminal agree on completing the transaction offline (based on both entities' risk management policies) the card returns a TC (Transaction Certificate) approving the transaction. If either card or terminal want to continue online, the card produces an ARQC (Authorization Request Cryptogram), which the terminal passes on to the bank in an *online authorization request*. If verification is successful, the bank returns an *authorization response* message containing Issuer Authentication Data (IAD) and possibly a command script to be delivered to the card. The terminal then issues the second GENERATE_AC command including the IAD and the command script.

ARQC, TC and IAD are authenticated using MACs (Message Authentication Codes). These are generated

by 64-bit block ciphers using a session key k derived from a master key shared by the card and the issuer. The issuer can verify both ARQC and TC; in the online case the card verifies the IAD in the second GENERATE_AC command and thereby authenticates the issuer's response. The terminal triggers the generation and verification of these cryptograms but cannot verify them.

4. EMV Payments in the Internet Scenario

In the remainder of this paper, we analyze if and how EMV cards can be used for secure Internet payments.

The scenario (Figure 4) depicts a customer using his or her EMV card for online purchases from a PC that has a card acceptance device (reader) attached to it. The merchant still acts as the EMV terminal, issuing and receiving EMV commands and responses, but now communicates with customer and bank over the Internet.

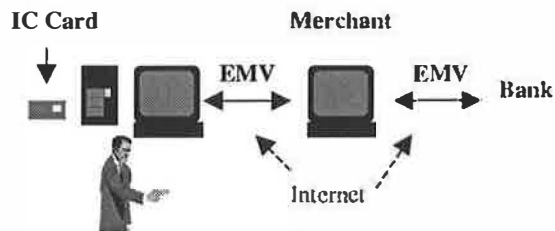


Figure 4. The EMV Internet Scenario

PIN verification deserves some special attention. While, in the POS scenario, the terminal secures the transaction by making sure the PIN is verified correctly (by card or bank), PIN verification in an Internet setting can and should no longer be controlled by the merchant:

1. *Online PIN verification* now requires the PIN to be sent from card to merchant to bank over insecure Internet connections. Even when encrypting (e.g., using SSL [5]) communication, the PIN appears in clear in the merchant's software, which is too high an exposure.
2. Even *offline PIN verification* (using VERIFY) can no longer be controlled by the merchant:
 - requiring VERIFY (including the PIN) to be issued by the merchant assumes the PIN first to be sent to the merchant over an Internet connection (and unnecessarily expose it);
 - furthermore, the merchant doesn't gain any security from the VERIFY result since it is not authenticated, and when received over the Internet, doesn't guarantee to the merchant that this was the PIN verification result produced by the card (or, stronger, that the card ever executed the VERIFY command!).

Consequently we recommend (and assume in the following discussion) in the Internet scenario:

- Only the offline PIN authentication mechanism (VERIFY by the card) to be used;
- The VERIFY command to be issued locally (at the cardholder terminal); and
- The card to actually enforce cardholder verification by only issuing ARQC/TC after a successful VERIFY. This is currently not an explicit condition in the EMV specifications; but since in our scenario, neither merchant nor bank can enforce cardholder verification, we now have to make this an explicit condition.

We now map the online and offline transaction flows of Figure 3 to the Internet scenario of Figure 4, resulting in

the 'online and offline EMV Internet flows' as depicted in Figure 5. In the following paragraph we analyze the security of these two scenarios by checking them against the security requirements defined in Section 1. Table 1 also summarizes the results of this analysis.

1. *Authorization customer to bank.* In both scenarios the transaction is weakly authorized to the bank by the customer who generates an ARQC and a TC using a key shared with the issuer.
2. *Authorization merchant to bank.* The merchant does not explicitly authorize the transaction in the above protocols; there is only an implicit authorization by asking the bank for an authorization (by sending ARQC) or clearing of the payment (by sending TC).
3. *Payment guarantee for merchant.* In both protocols the merchant does not obtain a sufficient guarantee for the payment which is desirable before delivering goods. The merchant neither receives an authorization of the transaction by the bank nor by the customer because it cannot verify any of TC, ARQC, or IAD.
4. *Authentication and certification of merchant to customer.* EMV does not provide mechanisms to authenticate the terminal and to certify the merchant. The merchant's terminal identifier TermID is included in both ARQC and IAD such that the customer does have a guarantee that only a merchant with the TermID in the ARQC can claim the payment. However, the customer has no way of linking the TermID to the merchant s/he thinks the payment is made to, such that merchant impersonation attacks cannot be excluded.
5. *Payment receipt for customer.* The evaluation of protocols with regard to this requirement depends on the definition of valid payments and the contract between the customer and the issuer (see A3 and A4). In the offline scenario the customer does not get any authenticated proof of the payment. In the online case, one might consider the IAD as a payment receipt from the bank. This assumes, in turn, that the bank's authorization response completes the payment and that the merchant can consider the ARQC together with the IAD as a guarantee for payment. This is unlikely since the merchant can neither verify the ARQC nor the IAD.

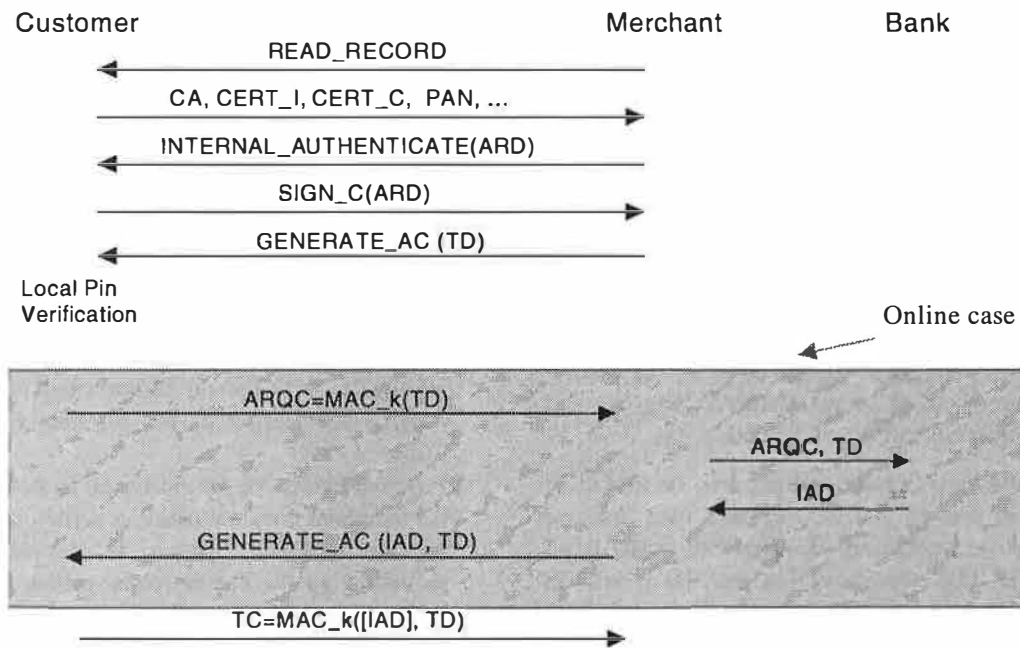


Figure 5. On-and offline EMV Internet Scenario

	Online	Offline
Part I. : GENERAL		
BANK		
1. authorization customer to bank	Y (weak)	Y (weak)
2. authorization merchant to bank	N	N
MERCHANT		
3. payment guarantee for merchant		
• authorization bank to merchant	N	N
• authorization customer to merchant	N	N
CUSTOMER		
4. merchant authentication + certification	N	N
5. payment receipt for customer		
• from the merchant	N	N
• from the bank	N	N
ALL PARTIES		
6. atomicity of payments	Y	Y
7. privacy, anonymity	N	N
Part II. SMARTCARD-SPECIFIC		
8. cardholder authorization	Y (if card-enforced VERIFY)	Y (if card-enforced VERIFY)

Table 1. Security Analysis of Online and Offline EMV Internet Scenarios
(Y = Requirement Satisfied; N= Requirement Not Satisfied)

6. *Atomicity of payments.* Atomicity of payments is provided in both protocols based on assumption A2 that money transfers between accounts are traceable.
7. *Privacy, anonymity* are not supported: EMV does not encrypt transaction data on the card-terminal channel, and the constant customer identification is present in the data read from the card. Privacy and anonymity issues are not further analyzed in the following.
8. *Cardholder authorization.* Based on assumption A5, this is achieved when using card-enforced PIN verification as recommended earlier in this section. Note that local PIN Verification in Figure 5 is shown to occur just before the card generates the ARQC; however it may be performed in an earlier stage (as long as the user is made aware of and agrees with the transaction data at the moment s/he enters the PIN).

The results of the analysis in Table 1 show a majority of unsatisfied requirements (N) without a distinction between offline and online scenarios. This is a result of the EMV specifications being developed for the POS scenario where the EMV terminal is under some control by the merchant (sometimes also bank), the purchase is performed face-to-face and merchant and bank communicate over secure connections. We shortly list and illustrate these assumptions.

1. A (physically) immediate and secure channel is assumed between card and terminal:
 - No tools for secure messaging between card and terminal are provided;
 - The result of PIN verification cannot be authenticated by the terminal;
 - Terminal applications rely upon correctness and integrity of other data returned by the card (e.g., static data authentication, risk management data).
2. A secure channel is assumed between terminal and bank:
 - no tools for secure messaging between terminal and bank are provided (e.g., online authorization messages are not authenticated between them).
3. The merchant is assumed to trust the bank:
 - the terminal cannot verify Application Cryptograms (ARQC, TC) or IAD.
4. The bank is assumed to trust the terminal to deliver messages to the card:
 - some security mechanisms rely upon the delivery of issuer messages to the card by the

terminal (e.g. command scripts in the authorization response).

5. The terminal is assumed not to be counterfeitable and not to be illegally manipulatable:
 - there is no explicit mechanism to authenticate the terminal to the card;
 - the terminal does not explicitly authorize/sign any part of the transaction;
 - the card does not obtain a payment receipt from the terminal.
6. The purchase is assumed to be performed face-to-face:
 - merchant authentication is not in the scope of EMV;
 - a guarantee for the delivery of goods is out of the scope of EMV;
 - a description of goods is not part of the transaction data.
7. It is assumed that the physical presence of the same card is verifiable during the transaction:
 - different parts of the transaction protocol are not explicitly linked;
 - there is no mechanism for the terminal to verify that the same card is used for different parts of the protocol (e.g. DDA, ARQC generation, TC generation).

Before discussing mechanisms which can increase the security of using EMV over the Internet, we summarize the most important vulnerabilities resulting from the above "N"s in the table. This is done to further illustrate possible risks and threats, and to point out their relative importance.

4.1. No payment guarantee for the merchant

This is probably the most serious problem: without a payment guarantee the merchant may lose money when delivering goods which are not paid for afterwards.

- *No bank to merchant authorization:* Since the merchant cannot verify the bank's authorization response (IAD), an attacker could impersonate the bank to the merchant with an invalid IAD, convincing the merchant the transaction was successful; alternatively, valid transaction data or a valid IAD can be modified during the transaction without the merchant being aware; or, the bank might repudiate the authorization afterwards.
- *No customer to merchant authorization:* This is especially critical in the offline case because the merchant has to accept a payment without being able to verify the TC. Anyone can make a payment

on the cardholder's behalf (though DDA would at least require the fraudster to have the card) or the cardholder can repudiate a payment s/he actually made. Even if a valid TC was issued by the card, it can be modified on the way to the merchant.

4.2. No merchant authorization

The absence of an explicit authorization of the transaction by the merchant means that an attacker may impersonate a real merchant to both customer and bank, and conduct a successful transaction on behalf of the real merchant who might not even be aware. This vulnerability can be exploited also by dishonest customers and merchants: a merchant can repudiate a transaction afterwards, claiming the above attack scenario has occurred; a dishonest customer may exploit the lack of merchant authorization by intercepting and modifying the transaction data on the merchant to bank channel. In the attack scenario as well as the dishonest merchant scenario, the customer does not get the ordered goods and has to claim refund while in the last scenario the merchant does not receive the expected payment for possibly delivered goods.

4.3. No merchant-to-customer authentication and certification

For debit or credit payments the danger for the customer caused by lack of merchant authentication is limited: the customer can only lose money to a legitimate merchant if we assume that the bank only clears payments for legitimate merchants. The absence of a merchant-to-customer (M-C) authentication mostly reinforces the danger caused by the absence of a merchant-to-bank (M-B) authorization, in the sense that a fully complete, normal and legitimate payment to M can take place without M being involved in any stage of the EMV protocol. On the contrary, a reasonable protection can be achieved if at least one of the two, M-C authentication or M-B authorization, is provided. Then either the customer or the bank can verify whether M is the merchant corresponding to the TermID in the ARQC or TC. If there is M-B authorization (at least during clearing), but no M-C authentication then it only remains critical that the customer might communicate with a different merchant than intended.

4.4. No receipt for the customer

Not receiving a payment receipt is mainly critical for the customer if s/he buys goods to conditions which change rapidly (e.g. stocks or shares). Especially in the offline protocol, the customer does not have any proof of having bought something to specified conditions before the actual clearing is performed and s/he has

received his or her statement of account. This can cause a loss of goods, opportunities, or money to the customer if the merchant denies certain conditions.

Note that within EMV it is impossible to simultaneously provide the merchant with a payment guarantee and the customer with a receipt because one message always has to be sent last. A simultaneous payment guarantee and customer receipt could be provided if the protocol were embedded in some *fair-exchange protocol* (such as [2]), which is out of the scope of EMV.

5. Mechanisms to Add Security when Using EMV over the Internet

We now discuss the benefits of different mechanisms which can secure EMV when used over the Internet. The protocol vulnerabilities of 'bare' EMV over the Internet relate to the absence of authorization of certain messages, and to the absence of authentication and certification of the merchant to the customer. We first analyze the merits of using a transport-layer mechanism such as SSL to secure the communication channels used, a solution which doesn't impose any changes on the EMV infrastructure. Given the limited improvements achieved with this approach, we then recommend some modifications to the EMV infrastructure that might allow a more secure use of EMV for Internet payments.

5.1. Securing communication channels

To the extent that SSL can provide initial authentication between communicating parties and integrity and/or confidentiality protection of the ensuing dialogue, it can provide a reasonable degree of protection against attacks by outsiders, under the condition that all parties involved adequately secure their systems. However, as discussed in the following paragraphs, SSL cannot provide the necessary authorizations we discussed before that are needed to protect parties against dishonest insiders.

SSL cannot authenticate individual EMV messages, rather it integrity-protects a data stream, which in addition could carry data generated by applications other than EMV. It secures the data using a shared session key which is temporary and cannot be tied to a specific party, except by its communication partner, and then only during the existence of the connection. Obviously, SSL 'authenticated' messages or data streams can never have any authenticating value to a third party, regardless of trust assumptions of this third party. (One could of course argue whether this is the case for EMV ARQCs or TCs. But given the assumed tamperproofness of the cards, and possibly certified

security of a bank's systems, EMV ARQCs or TCs may be considered by a third party as non-repudiable evidence.)

The authorizing value they have to the receiving party during the connection depends entirely on the receiver's trust in the sender's system and the sender's honesty. In a model where banks and merchants trust each other, this may suffice to add a weak authorization value to EMV messages exchanged between them; less clear is the authorization value for messages exchanged between customer and merchant. Specifically, in the offline scenario, it cannot provide a customer-authorized payment guarantee for the merchant.

Summarizing, we can say that SSL, under certain conditions, can add reasonable security against outsider attacks, but does not provide the authorization of EMV messages necessary to protect against dishonest insiders (or against honest insiders using insecure systems). In the next subsections, we suggest two modifications to EMV which can help towards solving these problems.

5.2. Signed authorization response

In the online scenario, an undeniable payment guarantee for the merchant may be provided by the (issuing) bank signing the authorization response message with its private signature key. The authorization response message becomes

SIGN_I (Y/N, Transaction Data, IAD)

where SIGN_I() stands for a signature with message recovery using the (issuer) bank's private signature key. This message can then be verified by the merchant (who already has obtained the issuer's public key during DDA) and can be submitted to the issuer again for final clearing.

The advantages of this extension are:

- This signature provides the merchant with an undeniable payment guarantee. Lack of a payment guarantee for the merchant was a major vulnerability in the above 'bare EMV' protocols (see 4.1).
- The signature prevents the Transaction Data (TD) from being modified during a protocol run without the merchant noticing it. Since the TD is included in the signature the merchant can refuse to deliver the goods if the TD is not correct. This simultaneously weakens the threats incurred by a missing authorization of the merchant to the bank (see 4.2).
- Since the merchant now has a payment guarantee before passing the IAD to the customer, the second

GENERATE_AC command may now be considered as a payment receipt for the customer – assuming at least that the customer gets the IAD from the merchant (and not directly from the bank!) (see 4.4).

- No further keys have to be distributed in addition to the ones already needed for DDA.
- The extension is possible with current cards which support DDA and therefore have stored the issuer's certificate CERT_I. Only slight modifications of the terminal specifications are required to accommodate the increased length of the data fields of the authorization response message.

A disadvantage of the approach as described above is that the issuer's private key – intended only to certify card public keys – is used more often and for other purposes, increasing its exposure. This is critical because the corresponding public key is stored on many cards and therefore hard to replace in case of a compromise. Therefore we recommend to use a separate key for signing authorization responses. Using a second issuer public key (and certificate CERT2_I) for this purpose is quite costly since it has either to be stored on (and read from) the card, or sent by the issuer to the merchant as part of the authorization response. A solution which combines security and low overhead can be provided by the acquirer signing the authorization response (as opposed to the issuer). Since the merchant has a long-term relationship with the acquirer, it can be assumed that the acquirer's public key is stored permanently by the merchant.

5.3. Public-key Transaction Certificate (TC)

Another proposed change to the EMV specifications is the use of a public-key signature also for TC generation. The TC becomes

TC = SIGN_C (TD, [IAD])

signed using the card's private key. A public key-based TC is verifiable by the merchant and can be considered as a payment guarantee depending on contract terms between merchant and acquirer (which may require certain risk management measures by the merchant). A public-key signed TC seems to be a natural extension given that DDA-capable cards already have the signature generation capability. However, in order to support this extension, message formats for cryptogram generation need to be changed, which may have a major impact on the whole EMV infrastructure and poses challenges related to backward compatibility.

Security gains that can be achieved by using a public-key based TC are:

- The merchant receives an undeniable authorization of the transaction by the customer and thus possibly (depending on contracts) a payment guarantee (see 4.1) also without online authorization;
- The transaction data cannot be modified without the merchant noticing it. This denies some of the threats incurred by a missing merchant-to-bank authorization (see 4.2).
- The TC can now also be considered an undeniable authorization customer-to-bank, as opposed to the weak authorization using the shared-key mechanism (see R1).

Despite the necessary changes to support a public key TC, we strongly recommend this extension to EMV since it is absolutely crucial to provide security in the offline case and therefore is a must when considering the use of EMV as a purse (e-cash) payment system in the Internet.

5.4. Merchant authentication

Changes proposed in 5.2 (online payments only) or 5.3 (especially important for offline payments) can greatly improve the security of the respective EMV-Internet scenarios. Vulnerabilities remain, primarily related to the lack of authentication and authorization of the merchant to both customer and bank. Closing these holes in a rigorous way by providing merchant authentication in EMV largely impacts the EMV infrastructure which currently does not allow for the storage of secret keys in merchant terminals. However, to the extent that the keys stored need not be system-wide symmetric keys but rather the merchant's own private signature key for authentication to bank and/or customer, we believe that such a modification can only strengthen overall security. Such a change first of all allows the merchant to sign the authorization request message, providing secure authorization by the requesting merchant. It also allows merchants to authenticate to the card and to deliver a signed payment receipt to the customer which in the offline case is the only means for the customer to get a receipt (other than an after-the-fact account statement). Since cards with signature verification capability are not likely to be used soon, the signature verification could be done in the trusted card reader (or eventually, in the PC software).

6. Related Work

The principle of using existing payment smart cards to secure Internet transactions has been applied in recent projects such as the e-COMM [4] and C-SET [6] projects in France. Both integrate shared-key based Transaction Certificates from existing EMV-like

banking cards within SET or SET-like protocols. In this paper, rather than proposing a specific solution, we have tried to give a comprehensive and systematic overview of the security features and limits of a variety of related solutions, and hope it can be applied in the evaluation or design of similar systems.

7. Conclusion

The use of EMV 'as is' over the Internet has major (and unacceptable) security shortcomings. Securing the communication channels between the different parties (customer, merchant, bank) using secure communication protocols can prevent mainly outsider attacks. However it does not solve the inherent lack of authentication in the EMV protocol. Therefore we propose a number of EMV extensions which can increase security in the Internet setting.

The most challenging is the EMV offline scenario, where only the use of a public-key based Transaction Certificate provides appropriate security to the merchant. This scenario is particularly important if EMV'96 is used for purse (e-cash) applications.

Online EMV authorization in an Internet setting, though currently insecure because of merchant as well as bank impersonation attacks, can be made more secure by digitally signing authorization requests and responses. Lack of initial authentication and certification of the merchant to the customer is a vulnerability only to be solved by extending the EMV infrastructure with terminal-to-card (alternatively, terminal-to-reader or terminal-to-user's PC) dynamic authentication. In the absence of terminal authentication, software-based mechanisms (e.g. SSL server-to-client authentication) can be put in place to thwart the biggest risks of outsider attacks.

8. Acknowledgment

The authors thank Michael Waidner for his helpful comments and suggestions.

9. References

- [1] N. Asokan, P. Janson, M. Steiner and M. Waidner, "The State of the Art in Electronic Payment Systems", in *IEEE Computer*, September 1997.
- [2] N. Asokan, V. Shoup and M. Waidner, "Asynchronous Protocols for Optimistic Fair Exchange", in *1998 IEEE Symposium on Research in Security and Privacy*, Oakland, May 1998, pages 86-99.

- [3] M. Bellare, J.A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, M. Waidner, "iKP - A Family of Secure Electronic Payment Protocols", in *First USENIX Workshop on Electronic Commerce*, July 1995, pages 89-106.
- [4] E-Comm, "The e-COMM Solution", 1998. <http://www.e-comm.fr/anglais/solution.html>.
- [5] T. Elgamal and K. Hickman, "The SSL Protocol (version 3)", Netscape Communications, Internet Draft, June 1995.
- [6] Europay, "Secure Trading over the Internet thanks to C-SET."
<http://www.europayfrance.fr/us/commerce/secure.html>.
- [7] Europay, Mastercard, Visa, "EMV'96 Integrated Circuit Card Specification for Payment Systems, Integrated Circuit Card Terminal Specifications for Payment Systems and Integrated Circuit Card Application Specification for Payment Systems", Version 3.1.1, May 1998.
- [8] Mastercard and Visa, "SET Secure Electronic Transactions Protocol, version 1.0. Book One: Business Specifications, Book Two: Technical Specification, Book Three: Formal Protocol Definition", May 1997. Available from <http://www.mastercard.com/set/#down>.
- [9] B. Pfizmann and M. Waidner, "Properties of payment systems - general definition sketch and classification", Research Report RZ 2823, IBM Research, May 1996.

Breaking Up Is Hard To Do: Modeling Security Threats for Smart Cards

Bruce Schneier
Counterpane Systems
schneier@counterpane.com

Adam Shostack
Netect, Inc.
adam@netect.com

February 5, 1999

Abstract

Smart card systems differ from conventional computer systems in that different aspects of the system are not under a single trust boundary. The processor, I/O, data, programs, and network may be controlled by different, and hostile, parties. We discuss the security ramifications of these “splits” in trust, showing that they are fundamental to a proper understanding of the security of systems that include smart cards.

1 Introduction

Smart cards, credit-card-sized devices with a single embedded chip—CPU and RAM/ROM—are viewed by some as “magic bullets” of computer security. They are being proposed (and used) for access control, electronic commerce, authentication, privacy protection, etc. Unfortunately, there is little analysis of the security risks particular to smart cards, and the unique threat environments that they face.

In this paper, we discuss the security model of a smart card system independently of its application. We look at the fundamental properties of a smart card—a CPU and memory device with no means of communicating with the outside world—and show how these properties make systems based on smart cards riskier than similar systems based on self-contained computers. A clear example is a person carrying a card whose computer is under someone else’s control. This is an unusual situation for a typical computer, and a common one for a smart card. We show that for many applications, using a smart card securely means understanding it not as a “trusted” computation platform, but as a data storage device with limited computational abilities.

1.1 From Computers to Smart Cards

The best way to understand the threats facing a smart card is to start with the threats associated with a conventional desktop computer. We believe that the most important security aspect of smart cards, as participants in protocols, is the way in which they differ from other computational devices. By starting with a general purpose computer, and splitting apart its various functions into those that make up a smart card and its operating environment, we can examine each change and how it affects security. Each of these splits adds opportunities for attack. For example, consider a case where the owner of a card does not control the data stored on it. This leads to attacks by the person possessing the card against the data stored within it. This attack simply isn’t possible if there is no such split.

Our model of a general purpose computer consists of a CPU, storage, input/output devices, and power supply. The CPU is the primary processor of the computer, responsible for carrying out computation. In a normal computer, it is tightly coupled to its storage, such as RAM, disk drives, or tape, as well as its generalized I/O devices, such as keyboards or mice for input, terminals or printers for output, and various digital communication ports, such as serial ports or Ethernet cards. In this configuration, the computer can be treated as a single unit for most threat models.

We begin by miniaturizing the computer, which adds nothing beyond a useful visualization tool. Consider a computer such as the REX personal organizer. This PC-CARD has a small screen, a PC-CARD interface to communicate with another computer, and a few buttons for input. We will now transform the REX, in stages, into a smart card, showing how each step of the transformation leads to new vulnerabili-

ties.

Consider the I/O port, and replace it with a slow-speed serial port. The system that the card connects to has a limited ability to attack it, since the card is presumably going to be attached only to its owner's computer, or perhaps, for a few moments, to another to trade contact information. The card, throughout, has the ability to send and receive information through its screen and buttons. It would not be difficult to transform something like the REX into a secure electronic checkbook. (There are other engineering challenges, but it is substantially easier than building the same system with a smart card.)

Continue by decoupling the input mechanism, such that user input must go through a keyboard attached to the reader. It is obvious that the keyboard could record PIN and card information for use in a later attack. Lastly, remove the screen, such that the card has no way of communicating with its user except through a screen of indeterminate fealty.

The essential characteristic of a smart card is that its functionality is split in ways unusual for a computer. These splits mean that a smart card is "handicapped," by which we mean "unable to interact with the world without outside peripherals." This is the essential nature of smart cards: one that differentiates them from portable computers such as the Palm Pilot, and that defines the trust model in which they are forced to operate. Other splits may and do occur, but the fundamental one is that of being restricted in their I/O.

Smart card functionality is split in other ways. The cardholder might not have any control of the software running on the card. In the case of multifunction cards, the card issuer might not have any control either. The owner of the data inside the card might not be the cardholder, and the data owner might require that the cardholder not be able to modify, or even view, the data.

In the following sections, we examine the ramifications of the split described above, as well as others commonly found in smart card systems. Our models often include five or six parties. We examine in depth how the parties, when split, might attack each other. We also examine the motivations that cause attackers to engage in the variety of mischief that becomes possible when roles are split apart. And finally, we discuss different resistance models.

2 Model Trust Environment of a Smart Card

There are many parties potentially involved in any smart card-based system. Usually, there are at least five or six, including the cardholder, the terminal, the data owner, the card issuer, the card manufacturer, and the software manufacturer.

- The **cardholder** is the party who has day to day possession of the smart card. The card is in his wallet; he decides whether and when to use it. In the case of a smart card used as an electronic wallet, he is the person to whom the wallet was issued. He may control the data on the card, depending on the system, but it is highly unlikely that he had control of the protocols, software, or hardware choices made in the creation of the card system. Note that this is in contrast to many personal computer-based systems, where the owner and user usually has some say in the system he is using.
- The **data owner** is the party who has control of the data within the card. In cases such as using a card as a mechanism for carrying digital certificates, the card owner is also the data owner. However, if the card is an electronic-cash card, the issuer of the cash is the data owner, and this split opens the possibility of attack.
- The **terminal** is the device that offers the smart card its interactions with the world. The terminal controls all I/O to and from the smart card: the keyboard by which any data is entered into the smart card, and the screen by which any data from the smart card is displayed. If the card is used as a phone calling card, this is the pay phone owner. If the card is used as an ATM identification card, this is the ATM service provider. If the card is a pay-TV membership card, the terminal is the set-top box.¹

¹The previous two examples—ATM identification card and pay-TV membership card—illustrate times when the terminal, as well as the smart card, may be broken into several parties. In the case of an ATM, the use of another bank's ATM network and terminal is common, which means that the bank cannot rely on the terminal to be friendly. In the case of the pay-TV system, the terminal is in the long-term possession of the user, and can be attacked in the safety and comfort of the user's home. In cases where the terminal ownership, programming, possession, or other functions are split, a full analysis needs to be performed to ensure that the security impacts of the splits are understood.

- The **card issuer** is the party who issued the smart card. This party controls the operating system running on the smart card, and any data that is initially stored on the smart card. If the card is a telephone payment card, the issuer is the phone company. If the card is an employee ID card, the issuer is the employer. Sometimes the issuer just issues the card and then disappears from the system; other times he is involved with the system throughout. In some multi-function cards, the card issuer may have nothing to do with the applications running on the card, and may only control the operating system. In other multi-function cards, the same issuer may control all the applications running on the card.

From the security analysis point of view, it is often simplest to view the card issuer, the manufacturer, and the software engineers as the same party; however, they rarely actually are. Hence:

- The **card manufacturer** is the party who produces the smart card. Note that this is a simplification; the manufacturer may or may not own the fabrication facility in which the chips are actually made; they may have subcontracted design functions, and they may be using third-party tools in their work, such as VHDL compilers. However, we model all of these as the card manufacturer. Opportunities to subvert the manufacture of the card come in many places, to a wide variety of individuals.
- The **software manufacturer** is the party who produces the software that resides on the smart card. This is again a simplification of a probably complex array of makers of compilers, utilities, etc. Issues of trusting trust [Tho84] arise here in the same ways they do with the card manufacturer.

3 Examples of Trust Splits in Smart Card Systems

Following are representative smart card-based systems, described in terms of what parties control different aspects of the system. This list is not meant to be exhaustive, and there are both other examples of splits described here and other splits not described here.

- **Digital Stored Value Card.** These are payment cards intended to be substitutes for cash. Both Mondex and VisaCash are examples of this type of system. The card owner is the customer. The terminal owner is the merchant. The data owner and the card issuer are both the financial institution that supports the system.
- **Digital Check Card** This is similar to the card above, except that the card owner is the data owner.
- **Prepaid Phone Card.** These are simply a special-use stored value card. The card owner is the customer. The terminal owner, data owner, and card issuer are all the phone company.
- **Account-based Phone Card.** In this system, the smart card does not store an account balance, but simply an account number which is a pointer into a back-end database. The card owner and data owner is the customer, while the terminal owner and card issuer is the phone company.
- **Access Token.** In this application, the smart card stores a key which is used in a login or authentication protocol. In the corporate case, the cardholder is the employee, and the data owner, terminal owner, and issuer are likely the company. In the case of a multi-use access token, the cardholder and data owner might be the same person, while the terminal owner may be a merchant and the data owner a financial institution.
- **Web Browsing Card.** In this application, a customer can use his card in his own PC to buy things on the WWW. This is another example of a cash card. The difference is that the cardholder and terminal owner are both the customer (i.e., the owner of the PC). The data owner and card issuer are both the financial institution.
- **Digital Credential Device.** In this application, the smart card stores digital certificates or other credentials for presentation to another party. Here, the cardholder and the data owner are both the same. The terminal owner is either the other party (in an in-store application, for example) or the cardholder (browsing on the WWW). The card issuer is the CA that issued the credentials, or some other party that collects the credentials.

- **Key Storage Card.** In this application, the user stores various (possibly verified) public keys in a smart card to protect them having to be stored on his less secure PC. Here, the cardholder, the data owner, and the terminal owner are the same.
- **Multi-Function Card.** This card is the most complicated. The card manufacturer and card issuer are separate, as are the software manufacturers. The data owner may be the cardholder for some applications, and a separate entity for others. There are multiple terminal owners, depending on which applications are on the card.

4 Smart Card Threat Models

An attack is simply defined as an attempt by one or more parties involved in a smart card transaction to cheat. We consider two classes of attackers, those who are parties to the system, and those who are interlopers. Attacks by participants could be a cardholder trying to cheat a terminal owner, a card issuer trying to cheat a cardholder, etc. Outsider attacks could be mounted by someone who steals a card: a temporary cardholder who steals a card from a legitimate cardholder, or replaces terminal software or hardware. Attacks by outsiders are often similar to attacks on protocols involving general purpose computers; however, they may take advantage of various properties of the system created by the separation of roles.

Motives for attack fall into a few broad categories [Sch97]. First and most obvious are financial thefts, including theft of money or credit, or theft of services sold to the general public, such as telephone cards. There are also impersonation attacks, where the card system is an intermediate target, with the system being attacked to gain access to some computer system, or other access control device. These differ from theft of service in that the user could not purchase the service legitimately. For example, the use of an access card to get into a computer system; computer access is generally available, but access to the particular system is the goal of the attacker. There are attacks on privacy, where one party wants more information than is given by the protocol. Lastly, there are publicity attacks, where the attacker is motivated not by any direct financial gain through attacking the system, but a desire for notoriety.

5 Classes of Attack

Due to the large number of parties involved in any smart card-based system, there are many classes of attacks to consider. Our goal here is to categorize them by function split. That is, we will look at attacks by system participants against one another. Most of these attacks are not possible in conventional computer systems, since they would take place within a traditional computer's security boundary. However, they are possible in the smart card world.

5.1 Attacks by the Terminal Against the Cardholder or Data Owner

These are the easiest attacks to understand. When a cardholder puts his card into a terminal, he is trusting the terminal to relay any input and output from the card accurately. For example, if a user puts a stored value card into a vending machine and makes a \$1 purchase, he is relying on the terminal to send a "deduct \$1" message to the card, and not a "deduct \$10." Similarly, when the card sends a message to the cardholder that says "balance = \$1," the cardholder is relying on the terminal's screen to relay that message accurately. The ability for a rogue terminal to do damage in this environment is significant, and it is impossible for the cardholder to detect this kind of fraud in the context of a single terminal. This kind of fraud has been attempted using fake ATM machines [?].

Prevention mechanisms in most smart card systems center around the fact that the terminal only has access to a card for a short period of time. Software on the card could limit the amount of damage a rogue terminal could do. A stored-value card could, for example, only allow the terminal to deduct \$1 maximum per transaction, and to perform no more than one transaction every minute [KS99]. However, there are prevention mechanisms that involve having the user own the smart card terminal, such as one attached to a personal computer. The real prevention mechanisms, though, have nothing to do with the smart card/terminal exchange; they are the back-end processing systems that monitor the card and terminals, and flag suspicious behavior.

5.2 Attacks by the Cardholder Against the Terminal

More subtle are attacks by the cardholder against the terminal. These involve fake or modified cards running rogue software, with the intent of subverting the protocol between the card and the terminal. For some examples, see [McC96].

Good protocol design mitigates the risk of these kinds of attacks, which can be made more difficult by hard-to-forge physical aspects of the card (e.g., the hologram on Visa and MasterCard cards), which can be checked by the terminal owner manually. Note that digital signatures on the software are not effective here since a rogue card can always lie about its signature, and there is no way for the terminal to peer inside the card. Defending against this kind of attack requires another function split: the cardholder must not be able to manipulate the data inside the card.

5.3 Attacks by the Cardholder Against the Data Owner

In many smart card-based commerce systems, data stored on that card must be protected from the cardholder. In some cases, the cardholder is not allowed to know that data. A building access card, for example, could have a secret value inside the card; knowledge of this value could allow the cardholder to make additional access cards. Or knowledge of a secret key in an electronic commerce card could allow the cardholder to make fraudulent transactions. In other cases, the cardholder is allowed to know the value, but not allowed to change it. If the card is a stored-value card, and the user can change the value, he can effectively mint money.

There are two essential characteristics of these attacks. One, the card must act as a secure perimeter, preventing the cardholder from accessing the data inside the card. In this context, the card may need to be fairly confident that it will detect and respond to attacks with a minimum of control over its environment. And two, the attacker has access to the card on his own terms. He is allowed to take the card into his laboratory and perform whatever experiments he wants to. He is allowed to take cards and destroy them in order to learn how they work.

There have been many successful attacks against the data inside a card. These attacks include reverse-engineering and defeating tamper-resistance

[AK96], fault analysis [BS97, BDL97], and side-channel attacks such as power and timing analysis [Koc96, Koc98b, KSWH98b, DLK+99].

These attacks have been particularly effective against pay-TV access cards [McC96, Row97], and have been used against digital cellular telephone access cards [BGW98]. They are starting to be used against stored-value cards for electronic commerce [Row97].

5.4 Attacks by the Cardholder Against the Issuer

There are many financial attacks that appear to be targeting the issuer, but this may be illusory. In fact, the attacks are targeting the integrity and authenticity of data or programs stored on the card. These attacks are made possible by the issuer's decision to use a smart card system where the cardholder holds data for the issuer or other party. Using the pay telephone application as an example, if the phone were to use an account-based system, where a simple card holds a very long account number that is used by the phone company to dereference an account stored on a back-end system, then there are account guessing and theft attacks based on the numbers. This sort of system can be enhanced by adding a challenge/response or inverted hash chain mechanism for sending replay resistant passwords. This makes strong use of a simple smart card in conjunction with a back office-managed authorization scheme to resist fraud. If the card issuer chooses to put bits that authorize use of the system in the card, they should not be surprised when those bits are attacked. These bits could be "authenticated" account numbers, or it could be a system with a key buried within the card, on the assumption that this key cannot be extracted, and proper completion of the protocol indicates that the card has not been tampered with. These systems all rest on the questionable assumption that the security perimeter of a smart card is sufficient for their purposes.

5.5 Attacks by the Cardholder Against the Software Manufacturer

Generally, in systems where the card is issued to an assumed hostile user, the assumption exists that the card will not have new software loaded onto it. This is enforced by the use of pre-issuance stages with

various one-way transformations being employed by the card manufacturer to ensure that the software is not tampered with. The underlying assumption may be that the split between card owner and software owner is unassailable, and relies on the separation being strong. However, attackers have shown a remarkable ability to get the appropriate hardware sent to them, often gratis, to aid in launching an attack.

5.6 Attacks by the Terminal Owner Against the Issuer

In a system closed to outsiders, such as some prepaid telephone cards, the terminal owner is also the card issuer (the phone company has both roles). In some more open systems, like Mondex, the terminal owner is the merchant and the card issuer is Mondex. The latter split introduces several new attacks.

The terminal controls all communication between the card and the card issuer (generally the back-end of the system). In this system, the terminal can always falsify records that have nothing to do with the smart card, refuse to record transactions, etc. The terminal can also fail to complete one or more steps of a transaction to facilitate fraud or create customer service difficulties for the issuer. By failing to complete the action of debiting a card, a terminal can cheat the issuer, or by completing a transaction and not offering service (i.e., a pay phone) can create a service nightmare.

These attacks are not related to the smart card nature of the system, and are simply attacks against the relationship between the terminal owner and the card issuer. Some systems try to mitigate this threat by having the card and back-end computer make a secure connection through the terminal. Many systems use monitoring on the back end to reduce the effectiveness of these attacks.

5.7 Attacks by the Issuer Against the Cardholder

In general, most systems presuppose that the card issuer holds the best interests of the cardholder at heart. This is not necessarily the case, and a malicious issuer can launch several attacks against cardholders.

These attacks are typically privacy invasions of one kind or another. Smart card systems that serve as a

substitute for cash must be designed very carefully to maintain the anonymity and unlinkability that are a property of cash money. Attacks or design failures can substantially reduce the privacy of the system. Alternately, a system may be sold as having more privacy than it in fact offers, allowing the issuer to gather data surreptitiously about the cardholders.

Features introduced into the card as the system matures may alter initial characteristics of the system with substantial impact on the privacy of the system. This can count as an attack by the issuer because the cardholder is rarely asked or able to discern the security impact of a change to the system made by the issuer. These changes are often not optional from the customer's viewpoint; the only choices are to accept the upgrade or leave the system. Lastly, this type of attack may be carried out by the issuer, or by the hardware or software designer, in collaboration with terminals, without the knowledge or consent of the issuer.

5.8 Attacks by the Manufacturer Against the Data Owner

Certain designs by manufacturers may have substantial and detrimental effects on the data owners in a system. The design of secure multi-user computers is a challenging one, and the security model to use to establish a secure kernel that offers processes protection from each other is not a solved problem. By providing an operating system that allows or even encourages multiple users to run programs on the same card, a number of new security issues are opened up.

The first, and most obvious, is subversion of the operating system and subsequently other programs. This is an area where mainstream operating system manufacturers have failed to provide adequate protection for the last thirty years. The vendors who have announced smart card operating systems recently do not have enviable records. However, even if the smart card operating system can be made secure, issues of user interface security remain and are exacerbated by the smart card's handicaps. How is the user (or the designer) to know what program is running when the card is inserted into a terminal? How to ensure that your program is talking to the terminal, and not through another program? How can a program that believes itself compromised terminate safely, and signal outward the cause for its demise? Or should it even try; what interesting attacks might become possible if a card announces its own imminent suicide? Can the card ensure that

once such a message is sent the action of destroying its memory is completed, in the presence of a possibly hostile power supply?

Less obvious would be intentionally poor random number generators [KSWH98a], or other aspects of cryptographic implementation that are difficult and arcane areas to test [Sch97, Sch98a, Koc98a, Sch98b]. The manufacturer is in an admirable position to engage in kleptographic attacks [YY96, YY97a, YY97b]. Of the major smart card vendors, none has an admirable record of creating operating systems that were free of exploitable vulnerabilities. In addition, by providing implementations of various supporting protocols, the vendor may be in a position to leak an application's keys using any of several subliminal channels [Sim84, Sim85, Sim86, Sim94].

And finally, it is possible for one application on a smart card to subvert another application running on the same smart card. It has been shown how to take a secure protocol and to create another protocol, also secure, such that the second protocol breaks the first protocol if both are running on the same device using the same keys [KSW96].

6 Transformative, or Impersonation, Attacks

There is a class of attacks based on separating or changing the roles played by various parties; for example, changing the cardholder by stealing the card may allow access to data that the cardholder has stored, or using ActiveX controls that allow an attacker to become (in essence) the terminal owner, engaging in the set of attacks available to terminal owners.

The essential character of a transformative attack is that a party is transformed, leading to an unexpected set of motivations for that party. When a card is stolen, the new cardholder (i.e., the thief) has lost all interest in maintaining the security of the account, and possibly in the physical integrity of the card. When a terminal is subverted, its desire to participate in a fair manner is replaced by a desire to subvert the protocol (why else subvert the terminal?). Thus, when a system assumes that the data stored on a card is secure because the interests of the cardholder and issuer are aligned, a vulnerability is opened by the theft of the card.

Alternately, we examine a system with a smart card reader attached to a PC, where that PC is acting

as part of the terminal. The terminal is presumed to be friendly to its owner; perhaps it is being used to carry Web certificates from home to work. Unfortunately, the terminal can be transformed by the introduction of an ActiveX control that changes the reader software. This attack, by changing the expected behavior of a component, can recast the security of the protocol. The behavioral change here can be active, in the case of changing a request and its associated display, or passive, in the case of monitoring attacks. Monitoring attacks can attack the privacy of the transactions made by the card or the secrecy of PIN or other data. The latter is probably a precursor to an active attack, not necessarily in the domain of the smart card protocol. That is, recall that PINs are often used in more than one system, and that the active attack does not need to attack the smart card system.

6.1 Attacks by Third Parties Using Stolen Cards

There are two differences between this attack and an attack by the cardholder. One, the thief does not have access to any secret information required to activate the card. And two, the thief has only a limited amount of time to carry out his attack before the cardholder will notice that his card has been stolen.

Hence, all the attacks by the cardholder are possible with the following addition: the thief is not concerned with any long-term repercussions against the legitimate cardholder. For example, a low-value stored-value card might deal with the potential of cardholder fraud by simply keeping records of cardholder transactions, and billing (or prosecuting) any discrepancies. A thief who steals a card would not be deterred by this defensive measure.

It is possible to build defenses into the system either at the card's or at the issuer's level. At the card level, there are perimeter and anomaly defenses available. The perimeter defense is that the card can consider several bad PIN attempts to be indicative of attack. (Note that this opens the card to a denial of service driven by a malicious terminal.) The anomaly detection defense would be for the card to store history information and detect a pattern change in its use. This is an aggressive requirement, but in those cases where a card can be used offline, it may make sense to raise a flag of some type, possibly requiring contact with its issuer before additional use to allow the back end system a chance

to make a more elaborate or sophisticated decision, or perhaps simply to defend the system against card duplication.

6.2 Eve and Mallet

If we assume that the use of a smart card is to allow protocol interactions between mutually distrusting parties, or at least parties whose interests diverge, then the protocols must resist the same set of attacks that they would if the systems were implemented with general purpose computers. Thus, most attacks based on eavesdropping or malicious protocol manipulation may be modeled as the case of one party attacking another. Assuming that the protocol is well designed, it will resist these attacks equally well if the attacker is internal or external.

6.3 Collaborative Attacks

Systems that rely on the split between various components being maintained as a hostile boundary without cooperation may find themselves surprised when roles they had thought split are brought together. The smart card and set top box, supposedly representing different interests, may collaborate in obtaining unauthorized service for the owner of the television. Similarly, the terminal's owner may be surprised to discover that both the card and the terminal, made and programmed by the same shop, have certain undocumented features. The number of possible collaborations and interesting models for attack grows with the number of parties to the system. Those who forget that most attacks are perpetrated by insiders will likely be reminded (assuming their fraud detection models are good enough.)

7 Resistance Models

There are, broadly, two ways to resist attacks against smart card systems. The first is to make specific attacks harder: use strong cryptographic protocols, increase tamper-resistance, etc. We don't discuss these methods in detail; we believe they are less effective and more prone to implementation and design failure than the second, which is to make entire classes of attack ineffective. This can be done most effectively by reducing the number of parties, or increasing the transparency of a party's role to the point where carrying out an attack is difficult. The easiest way to reduce the number of parties

is to combine roles so that there are fewer hats to wear. If, for example, the cardholder is also the data owner, all attacks by the cardholder against the data owner are simply irrelevant. Or, if the terminal owner is also the issuer, then attacks by the terminal owner on the issuer are only possible in the transformative case, where an attacker takes control of the terminal.

7.1 Fewer Splits

Each time a system has the design role of two or more parties merged into one, the avenues of attack that are available to one of those parties against the other disappears. For example, if the cardholder and terminal are merged by adding screen and data entry to the card, then the keysniffing and untrusted display problems simply disappear.

Contrariwise, adding parties to the system opens new venues of attack which need to be considered. The separation of the terminal and card from each other creates a venue which could scarcely have been designed better to enable man-in-the-middle attacks. The combination of physical encasement of the card, and terminal's control of the user interface and network allow most any such attack documented to be carried out if the protocol is not designed to handle it. Experience has shown that even many security products are released without consideration given to MITM, replay, and reflection style attacks. [Sho96, Sho97] Even if these attacks are considered, the addition of parties to a transaction makes managing keys, nonces, sequence numbers, and other defenses substantially more difficult.

Considering the smart card's inability to communicate with the outside world, the simplest reduction is to ensure that the cardholder and data owner are one. This is also usually one of the least expensive. The other extremely effective change to be made, adding screen and input devices to the card, involves a substantial increase in the cost of the card.

7.2 More Transparency

It is widely understood by the security community that the best way to ensure the security of a system is to allow widespread public examination of it. It has been shown repeatedly that interested attackers will obtain specifications or attack the system without them [Sho96, Bla94], and that open publication leads to review and analysis. (Examples are

IPSec, PGP, and S/MIME.) Combining the mechanisms of simplicity and openness greatly simplifies the task of reviewers who choose to examine a system. Thus, reducing the number of parties not only eliminates entire classes of attacks as shown above, but it also makes the task of analyzing the system simpler. The simplicity of the security analysis will likely cause the analysis to occur sooner, as well as giving it a higher likelihood of success.

The transparency defense involves cleanly separating roles so that attacks are more difficult to execute. For example, the Mondex system includes a variety of terminal types (some portable) that allow a user to check certain parameters independent of a merchant terminal. This allows a class of attacks on the cardholder or be discovered much more quickly. Access to the full set of Mondex stored parameters (i.e., the data owner's data) would presumably make the system that much more secure by increasing the audit-ability of the system. Similarly, an attack by the software manufacturer is made more difficult by the presence of strong and clear specifications, and/or open source implementations.

7.3 Design for Security

This defensive model of design is focused on designing systems to be secure from the architecture down [SSS+98]. Adding security to a system after the design phase has been shown to be difficult, expensive, and failure prone. Therefore, we offer a model where careful design from the start eliminates the need for many costly and complex attempts to bolt security on at a later phase. The reductionist model not only simplifies the process of design and implementation, but is fairly difficult to implement incorrectly. We have seen that implementation failures are a primary cause of cryptosystem failure in the field [And94, Sch97, Sch98a, Koc98a, Sch98b].

Another facet to the transparency defense is to avoid the complexities and risk of multi-application smart cards. Not using a multi-application smart cards both reduces the number of parties involved and creates a simpler operating environment with less complexity and potential for bugs. The reduction in the number of parties using the card (from N to 2) means that the issues of OS subversion and cross application attacks are practically eliminated.

8 Conclusions

We have shown that the splitting of the security perimeter is a difficult task. In particular, having a user carry a computer on behalf of a data owner he may wish to attack is a very risky situation for the data owner. We have also shown that the card's handicap of being unable to communicate makes it highly vulnerable to attacks by the terminal. These vulnerabilities are part of smart card systems by design, and require substantial effort to combat.

We have outlined a pair of fundamental defenses for cards, that operate at the system design level, offering system designers a new model in which to evaluate their systems. This model encourages pushing security into the earliest phases of system design. We offer as a prime candidate for improvement placing the user interface under the control of the user. System designs that re-combine the roles into more capable systems will likely find their investment results in fewer points of weakness.

References

- [And94] R. Anderson, "Why Cryptosystems Fail," *Communications of the ACM*, v. 37, n. 11, Nov 1994, pp. 32–40.
- [AK96] R. Anderson and M. Kuhn, "Tamper Resistance – A Cautionary Note," *Second USENIX Workshop on Electronic Commerce Proceedings*, USENIX Press, 1996, pp. 1–11.
- [BDL97] D. Boneh, R.A. Demillo, R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Advances in Cryptology—EUROCRYPT '97 Proceedings*, Springer-Verlag, 1997, pp. 37–51.
- [BGW98] M. Briceno, I. Goldberg, D. Wagner, "Attacks on GSM Security," work in progress.
- [BS97] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Advances in Cryptology—CRYPTO '97 Proceedings*, Springer-Verlag, 1997, pp. 513–525.
- [Bla94] M. Blaze, "Protocol Failure in the Encrypted Encryption Standard," *Pro-*

- ceedings of Second ACM Conference on Computer and Communications Security, ACM Press, 1994.
- [DLK+99] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestre, J.-J. Quisquater, and J.-L. Willerns, "A Practical Implementation of the Timing Attack," *CARDIS '98 Proceedings*, Springer-Verlag, 1999, to appear.
- [Jon93] K. Johnson, "One Less Thing to Believe in: High-Tech Fraud at an ATM," *The New York Times*, 13 May 93, pp. 1,B9.
- [Koc96] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Advances in Cryptology—CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 104–113.
- [Koc98a] P. Kocher, "Hidden Flaws: Avoiding Unexpected Weaknesses," *The 1998 RSA Data Security Conference Proceedings*, RSA Data Security, Inc., 1998.
- [Koc98b] P. Kocher, "Differential Power Analysis," available online from <http://www.cryptography.com/dpa/>.
- [KS99] J. Kelsey and B. Schneier, "Authenticating Secure Tokens Using Slow Memory Access," in preparation.
- [KSW96] J. Kelsey, B. Schneier, and D. Wagner, "Protocol Interactions and the Chosen Protocol Attack," *Security Protocols, International Workshop April 1997 Proceedings*, Springer-Verlag, 1998, pp. 91–104.
- [KSWH98a] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic Attacks on Pseudorandom Number Generators," *Fast Software Encryption, 5th International Workshop Proceedings*, Springer-Verlag, 1998, pp. 168–188.
- [KSWH98b] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side Channel Cryptanalysis of Product Ciphers," *ESORICS '98 Proceedings*, Springer-Verlag, 1998, pp. 97–110.
- [McC96] J. McCormac, *European Scrambling Systems*, Waterford University Press, 1996.
- [Row97] T. Rowley, "How to Break a Smart Card," *The 1997 RSA Data Security Conference Proceedings*, RSA Data Security, Inc., 1997.
- [Sch97] B. Schneier, "Why Cryptography is Harder than it Looks," *Information Security Bulletin*, v. 2, n. 2, March 1997, pp. 31–36.
- [Sch98a] B. Schneier, "Security Pitfalls in Cryptography," *CardTech/SecureTech Conference Proceedings, Volume 1: Technology*, CardTech/SecureTech, Inc., 1998, pp. 621–626.
- [Sch98b] B. Schneier, "Cryptographic Design Vulnerabilities," *IEEE Computer*, v. 31, n. 9, September 1998, pp. 29–33.
- [Sho96] A. Shostack, "Observed Weaknesses in the Security Dynamics Client/Server Protocol," *Network Threats Workshop, Dec 2-4 1996*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 38, R.N. Wright and P.G. Neumann, eds., American Mathematical Society, 1996.
- [Sho97] A. Shostack, "Low Hanging Fruit: A Replay Attack on the TIS FWTK," presentation at the CRYPTO '97 rump session.
- [SSS+98] C. Salter, O. Saydjari, B. Schneier, and J. Wallner, "Toward a Secure System Engineering Methodology," *New Security Paradigms Workshop 1998 Proceedings*, IEEE Computer Society Press, to appear.
- [Sim84] G.J. Simmons, "The Prisoner's Problem and the Subliminal Channel," *Advances in Cryptology: Proceedings of CRYPTO '83*, Plenum Press, 1984, pp. 364–378.
- [Sim85] G.J. Simmons, "The Subliminal Channel and Digital Signatures," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 364–378.

- [Sim86] G.J. Simmons, "A Secure Subliminal Channel (?)" *Advances in Cryptology: Proceedings of CRYPTO 85*, Springer-Verlag, 1986, pp. 33–41.
- [Sim94] G.J. Simmons, "Subliminal Channels: Past and Present," *European Transactions on Telecommunications*, v. 4, n. 4, 1994, pp. 459–473.
- [Tho84] Ken Thompson, "Reflections on Trusting Trust," *Communications of the ACM* Vol. 27, No 8, August 1984, pp. 761–763.
- [YY96] A. Young and M. Yung, "The Dark Side of Black Box Cryptography," *Advances in Cryptology — CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 89–103.
- [YY97a] A. Young and M. Yung, "Kleptography: Using Cryptography against Cryptography," *Advances in Cryptology — EUROCRYPT '97 Proceedings*, Springer-Verlag, 1997, pp. 62–74.
- [YY97b] A. Young and M. Yung, "The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems," *Advances in Cryptology — CRYPTO '97 Proceedings*, Springer-Verlag, 1997, pp. 264–276.

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

SAGE, the System Administrators Guild

The System Administrators Guild, a Special Technical Group within the USENIX Association, is dedicated to the recognition and advancement of system administration as a profession. To join SAGE, you must be a member of USENIX.

Member Benefits:

- Free subscription to *login*, the Association's magazine, published eight-ten times a year, featuring technical articles, system administration tips and techniques, practical columns on Perl, Java, Tcl/Tk, and C++, book and software reviews, summaries of sessions at USENIX conferences, and Snitch Reports from the USENIX representative and others on various ANSI, IEEE, and ISO standards efforts.
- Access to papers from the USENIX Conferences and Symposia, starting with 1993, via the USENIX Online Library on the World Wide Web.
- Discounts on registration fees for the annual, multi-topic technical conference, the System Administration Conference (LISA), and the various single-topic symposia addressing topics such as object-oriented technologies, security, operating systems, electronic commerce, and NT—as many as twelve technical meetings every year.
- Discounts on the purchase of proceedings and CD-ROMs from USENIX conferences and symposia and other technical publications.
- The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers.
- Discount on BSDI, Inc. products.
- Discount on all publications and software from Prime Time Freeware.
- Savings (10-20%) on selected titles from Academic Press, Morgan Kaufmann, New Riders/Cisco Press/MTP, O'Reilly & Associates, OnWord Press, The Open Group, Sage Science Press, and Wiley Computer Publishing.
- Special subscription rate for *The Linux Journal*, *The Perl Journal*, *IEEE Concurrency*, and all Sage Science Press journals.

Supporting Members of the USENIX Association:

C++ Users Journal	Internet Security Systems, Inc.	Performance Computing
Cirrus Technologies	Microsoft Research	Questra Consulting
Cisco Systems, Inc.	Motorola Australia Software Centre	Sendmail, Inc.
CyberSource Corporation	NeoSoft, Inc.	TeamQuest Corporation
Deer Run Associates	New Riders Press	UUNET Technologies, Inc.
Hewlett-Packard India	Nimrod AS	Windows NT Systems Magazine
Software Operations	O'Reilly & Associates Inc.	WITSEC, Inc.

Sage Supporting Members:

Atlantic Systems Group	Mentor Graphics Corp.	SysAdmin Magazine
Collective Technologies	Microsoft Research	Taos Mountain
D. E. Shaw & Co.	MindSource Software Engineers	TransQuest Technologies, Inc.
Deer Run Associates	Motorola Australia Software Centre	Unix Guru Universe (UGU)
ESM Services, Inc.	New Riders Press	
Global Networking	O'Reilly & Associates Inc.	
& Computing, Inc.	Remedy Corporation	

For further information about membership, conferences or publications, contact:

USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA.

Phone: 510-528-8649. Fax: 510-548-5738.

Email: office@usenix.org.

URL: <http://www.usenix.org>.

